The RWTH HPC-Cluster User's Guide Version 7.2

Release: November 2010
Build: November 11, 2010

Dieter an Mey, Christian Terboven, Paul Kapinos, Dirk Schmidl, Christian Iwainsky, Thomas Reichstein, Sandra Wienke, Michael Wirtz, Tim Cramer

Rechen- und Kommunikationszentrum der RWTH Aachen
(Center for Computing and Communication, RWTH Aachen University)

{anmey|terboven|kapinos|schmidl|reichstein|iwainsky|wienke|wirtz|cramer}@rz.rwth-aachen.de

What's New

The cluster underwent several changes since the last major version (6.6) of the RWTH HPC-Cluster User's Guide:

- On 01.01.2010, the RZ-ServiceDesk started its operations as the central and the only contact point (Single Point of Contact: SPoC) for IT services at RWTH Aachen.
- The old Opteron based Sun Fire v40z setup was turned off.
- Due to the very low usage, we decided to discontinue the official support for the Sun Solaris operating system. The remaining handful of computers running this last Unix species within the cluster will still be available until 2010, but without full support. Requiescat in pace, Solaris!

These topics were added or changed significantly[1] compared to the prior major release of this primer:

- Due to the new policy, the RZ-ServiceDesk is now the Single Point of Contact. Subsequently, all functional email addresses (e.g. hpc@rz.rwth-aachen.de) have been rerouted. Please send all feedback and questions to servicedesk@rz.rwth-aachen.de from now on.
- Added information about the Lustre file system. See chapter 4.3.2 on page 23.
- Added information about the AMD Barcelona processors. See chapter 2.5 on page 15.
- Added short description of PAPI. See chapter 8.7 on page 77.
- Added short description of Intel PTU. See chapter 8.2 on page 74.
- Rewritten section about NAG libraries. See chapter 9.6 on page 87.
- Numerous chapters regarding the Solaris operating system were removed.
- Information about the old AMD Opteron processors was removed.
- Added short description of Allinea Distributed Debugging Tool (**ddt**). See chapter 7.3.6 on page 68.
- On January 27, 2010, Sun Microsystems, Inc. was acquired by Oracle Corporation for US$7.4 billion, based on an agreement signed on April 20, 2009. As a consequence, the names of several former Sun products were changed:
  - Sun Studio → Oracle Solaris Studio
  - Sun HPC ClusterTools (CT) → Oracle Message Passing Toolkit
  - Sun Grid Engine (SGE) → Oracle Grid Engine

  We decided to adopt the new naming schema and to use "Oracle" whenever possible instead of "Sun", even though not all official names are known by now.

  The new name for Studio is unfortunate, since there (still) is a Linux version, too. Thus, and to not be riotous, we speak of just "Oracle Studio" in this document, even though we still mean "Oracle Solaris Studio". Similarly we speak of "Oracle MPI" when we mean "Oracle Message Passing Toolkit".

  Due to technical reasons we also decided *not* to rename the Oracle MPI module - it will continue to be called **sunmpi** so no changes in user scripts are necessary.

[1] The last changes are marked with a change bar on the border of the page

Table of Contents

1 Introduction

The Center for Computing and Communication of the RWTH Aachen University (Rechen- und Kommunikationszentrum (RZ) der Rheinisch-Westfälischen Technischen Hochschule (RWTH) Aachen) has been operating a UNIX cluster since 1994 and supporting Linux since 2004 and Windows since 2005. Today 95% of the cluster nodes run Linux, while Windows becomes increasingly popular.

The cluster is operated to serve the computational needs of researchers from the RWTH Aachen University and other universities in North-Rhine-Westphalia. This means that every employee of one of these universities may use the cluster for research purposes. Furthermore, students of the RWTH Aachen University can get an account in order to become acquainted with parallel computers and learn how to program them.[2]

This primer serves as a practical introduction to the HPC-Cluster. It describes the hardware architecture as well as selected aspects of the operating environments and the programming environment and also provides references for further information. It gives you a quick start in using the HPC-Cluster at the RWTH Aachen University including systems hosted for institutes which are integrated into the cluster.

If you are new to the HPC-Cluster we provide a 'Beginner's Introduction' in appendix B on page 99, which may be useful to do the first steps.

1.1 The HPC-Cluster

The architecture of the cluster is heterogeneous: The system as a whole contains a variety of hardware platforms and operating systems. Our goal is to give users access to specific features of different parts of the cluster while offering an environment which is as homogeneous as possible. The cluster keeps changing, since parts of it get replaced by newer and faster machines, possibly increasing the heterogeneity. Therefore, this document is updated regularly to keep up with the changes.

The HPC-Cluster mainly consists of Intel Xeon-based 4- to 24-way SMP nodes and a few AMD Opteron "Barcelona"-based 16-way SMP nodes. The nodes are either running Linux or Windows; a complete overview is given in table 2.3 on page 14.
Thus, the cluster provides two different platforms: Linux (denoted as *Lin*) and Windows (denoted as *Win*).

Accordingly, we offer different frontends into which you can log in for interactive access. See table 1.1 on page 8.

Frontend name	short alias	Proc - OS
cluster.rz.RWTH-Aachen.DE	cl	Linux
cluster-linux.rz.RWTH-Aachen.DE	cl-lin	Linux
cluster-linux-nehalem.rz.RWTH-Aachen.DE	cl-lin-n	Nehalem-Linux
cluster-linux-opteron.rz.RWTH-Aachen.DE	cl-lin-o	Opteron-Linux
cluster-linux-xeon.rz.RWTH-Aachen.DE	cl-lin-x	Xeon-Linux
cluster-x.rz.RWTH-Aachen.DE **cluster-x2**.rz.RWTH-Aachen.DE		Xeon-Linux (NX Software)
cluster-windows.rz.RWTH-Aachen.DE	cluster-win	Windows
cluster-win2003.rz.RWTH-Aachen.DE		Windows 2003

Table 1.1: Frontend nodes

To improve the cluster's operating stability, the frontend nodes are rebooted weekly, typically on Sundays. All the other machines are running in non-interactive mode and can be

[2] see appendix B on page 99 for a quick introduction to the Linux cluster

used by means of batch jobs (see chapter 4.5 on page 27).

1.2 Development Software Overview

A variety of different development tools as well as other ISV[3] software is available. However, this primer focuses on describing the available software development tools. Recommended tools are highlighted in **bold blue**.

An overview of the available compilers is given below. All compilers support serial programming as well as shared-memory parallelization (autoparallelization and OpenMP):

- **Intel (F95/C/C++)$^{\textbf{Lin,Win}}$**
- Sun Studio (F95/C/C++)$^{\text{Lin}}$
- **MS Visual Studio (C++)$^{\textbf{Win}}$**
- GNU (F95/C/C++)$^{\text{Lin}}$
- PGI (F95/C/C++)$^{\text{Lin}}$

For Message Passing (MPI) one of the following implementations can be used:

- **Sun MPI$^{\textbf{Lin}}$**
- Intel MPI$^{\text{Lin,Win}}$
- OpenMPI$^{\text{Lin}}$
- **Microsoft MPI$^{\textbf{Win}}$**

Table 1.2 on page 10 gives an overview of the available debugging and analyzing / tuning tools.

1.3 Examples

To demonstrate the various topics explained in this user's guide, we offer a collection of example programs and scripts.

The *example scripts* demonstrate the use of many tools and commands. Command lines, for which an example script is available, have the following notation in this document:
$ $PSRC/pex/100|| `echo ‘‘Hello World’’`

You can either run the script **$PSRC/pex/100** to execute the example. The script includes all necessary initializations. Or you can do the initialization yourself and then run the command after the "pipes", in this case **echo "Hello World"**. However, most of the scripts are offered for Linux only.

The *example programs*, demonstrating e.g. the usage of parallelization paradigms like OpenMP or MPI, are available on a shared cluster file system. The environment variable **$PSRC** points to its base directory. On our Windows systems the examples are located on drive **P:**.

The code of the examples is usually available in the programming languages C++, C and FORTRAN (F). The directory name contains the programming language, the parallelization paradigm, and the name of the code, e.g. the directory **$PSRC/C++-omp-pi** contains the Pi example written in C++ and parallelized with OpenMP. Available paradigms are:

- **ser** : Serial version, no parallelization. See chapter 5 on page 36

[3]Independent Software Vendor. See a list of installed products: http://www.rz.rwth-aachen.de/go/id/ond/

	Tool	Ser	ShMem	MPI
Debugging	**TotalView** Lin	**X**	**X**	**X**
	Allinea ddt Lin	X	X	X
	Sun IDE Lin	X	X	
	Sun dbx Lin	X	X	
	GNU gdb Lin	X		
	Intel idb Lin	X	X	
	MS Visual Studio Win	**X**	**X**	**X**
	PGI pgdbg	X		
	Sun Thread Analyzer Lin		**X**	
	Intel Thread Checker Lin,Win		**X**	
Analysis / Tuning	**Sun Performance Analyzer** Lin	**X**	**X**	**X**
	GNU gprof Lin	X		
	Intel VTune and PTU Lin,Win	**X**	**X**	
	Intel Thread Profiler Lin,Win		**X**	
	Intel Trace Analyzer and Collector Lin,Win			**X**
	Acumem Lin	**X**	**X**	
	Vampir Lin			**X**
	Scalasca Lin			X
	Marmot Lin			X

Table 1.2: Development Software Overview. Ser = Serial Programming; ShMem = Shared memory parallelization: Autoparallelization or OpenMP; MPI=Message Passing

- **aut** : Automatic parallelization done by the compiler for shared memory systems. See chapter 6.1 on page 53
- **omp** : Shared memory parallelization with OpenMP directives. See ch. 6.1 on page 53
- **mpi** : Parallelization using the message passing interface (MPI). See ch. 6.2 on page 59
- **hyb** : Hybrid parallelization, mixing MPI and OpenMP. See ch. 6.3 on page 63

The example directories contain Makefiles for Linux and Visual Studio project files for Windows. Furthermore, there are some more specific examples in project subdirectories like *vihps*.

You have to copy the examples to a writeable directory before using them. On Linux, you can copy an example to your home directory by changing into the example directory with e.g.
```
$ cd $PSRC/F-omp-pi
```
and running
```
$ gmake cp
```
After the files have been copied, a new shell is started and instructions on how to build the example are given.
```
$ gmake
```
will invoke the compiler to build the example program and then run it.

Additionally, we offer a detailed beginners introduction for the Linux cluster as an appendix (see chapter B on page 99). It contains a step-by-step description about how to build and run a first program and should be a good starting point in helping you to understand many topics explained in this document. It may also be interesting for advanced Linux users who are new to our HPC-Cluster to get a quick start.

1.4 Further Information

Please check our web pages for more up-to-date information:
http://www.rz.rwth-aachen.de/hpc/

The latest version of this document is located here:
http://www.rz.rwth-aachen.de/hpc/primer/

News, like new software or maintenance announcements about the HPC-Cluster, is provided through the **rzcluster** mailing list. Interested users are also invited to join the **rzcluster** discussion list at
http://mailman.rwth-aachen.de/mailman/listinfo/rzcluster

The mailing list archive is accessible at
http://mailman.rwth-aachen.de/pipermail/rzcluster

Please feel free to send feedback, questions or problem reports to

servicedesk@rz.rwth-aachen.de

Have fun using the RWTH Aachen HPC-Cluster!

2 Hardware

This chapter describes the hardware architecture of the various machines which are available as part of the RWTH Aachen University's HPC-Cluster.

2.1 Terms and Definitions

Since the concept of a processor has become increasingly unclear and confusing, it is necessary to clarify and specify some terms.[4] Previously, a processor socket was used to hold one processor chip[5] and appeared to the operating system as one logical processor. Today a processor socket can hold more than one processor chip. Each chip usually has multiple cores. Each core may support multiple threads simultaneously in hardware. It is not clear which of those should be called a processor, and everybody has another opinion on that. Therefore we try to avoid the term processor for hardware and will use the following more specific terms.

A *processor socket* is the foundation on the main board where a *processor package*[6], as delivered by the manufacturer, is installed. An 8-socket system, for example, contains up to 8 processor packages. All the logic inside of a processor package shares the connection to main memory (RAM).
A *processor chip* is one piece of silicon, containing one or more processor cores. Although typically only one chip is placed on a socket (processor package), it is possible that there is more than one chip in a processor package (e.g. there are two in the Pentium D "Presler"). A *processor core* is a standalone processing unit, like the ones formerly known as "processor" or "CPU". One of today's cores contains basically the same logic circuits as a CPU previously did. Because an n-core chip consists, coarsely speaking, of n replicated "traditional processors", such a chip is theoretically, memory bandwidth limitations set aside, n times faster than a single-core processor, at least when running a well-scaling parallel program. Several cores inside of one chip may share caches or other resources, but it is impossible to access data directly or hardware on one core from another one.

A slightly different approach to offer better performance is *hardware threads* (Intel: *Hyper Threading*). Here, only parts of the circuits are replicated and other parts, usually the computational pipelines, are shared between threads. These threads run different instruction streams in pseudo-parallel mode. The performance gained by this approach depends much on hardware and software. Processor cores not supporting hardware threads can be viewed as having only one thread.

From the operating system's point of view every hardware thread is a *logical processor*. For instance, a computer with 8 sockets, having installed dual-core processors with 2 hardware threads per core, would appear as a 32 processor ("*32-way*") system.[7] As it would be tedious to write "logical processor" or "logical CPU" every time when referring to what the operating system sees as a processor, we will abbreviate that.

Anyway, from the operating system's or software's point of view it does not make a difference whether a multicore or multisocket system is installed.

2.1.1 Non-Uniform Memory Architecture

For performance considerations the architecture of the computer is crucial especially regarding memory connections. All of today's modern multiprocessors have a non-uniform memory access (*NUMA*) architecture: parts of the main memory are directly attached to the processors.

Today, all common *NUMA* computers are actually *cache-coherent NUMA* (or *ccNUMA*) ones: There is special-purpose hardware (or operating system software) to maintain the cache

[4] Unfortunately different vendors use the same terms with various meanings.
[5] A chip is one piece of silicon, often called "die".
[6] Intel calls this a processor
[7] The term "n-way" is used in different ways. For us, n is the number of logical processors which the operating system sees.

coherence. Thus, the terms *NUMA* and *ccNUMA* are very often used as replacement for each other. The future development in computer architectures can lead to a rise of non-cache-coherent NUMA systems. As far as we only have ccNUMA computers, we use ccNUMA and NUMA terms interchangeably.

Each processor can thus directly access those memory banks that are attached to it (*local memory*), while accesses to memory banks attached to the other processors (*remote memory*) will be routed over the system interconnect. Therefore, accesses to local memory are faster than those to remote memory and the difference in speed may be significant. When a process allocates some memory and writes data into it, the default policy is to put the data in memory which is local to the processor first accessing it (*first touch*), as long as there is still such local memory available.

To obtain the whole computing performance, the application's data placement and memory access pattern are crucial. Unfavorable access patterns may degrade the performance of an application considerably. On NUMA computers, arrangements regarding data placement must be done both by programming (accessing the memory the "right" way; see chapter 6.1.2 on page 55) and by launching the application (*Binding*,[8] see chapter 3.1.1 on page 17).

2.2 Configuration of HPC-Cluster

Table 2.3 on page 14 lists all the nodes of the HPC-Cluster. The node names reflect the operating system running. The list contains only machines which are dedicated to general usage. In the course of the proceeding implementation of our integrative hosting concept[9] there are a number of hosted machines that sometimes might be used for batch production jobs. These machines do not show up in the list.

The Center for Computing and Communication's part of the HPC-Cluster has a peak performance of about 30 TFlops. Hosted systems have a peak performance of about 30 TFlops.

2.3 The Intel Xeon "Nehalem" based Machines

The Intel Xeon X5570 processors (Codename "Nehalem EP") are Quadcore Processors where each core can run 2 hardware threads (hyperthreading). Each core has a L1 and a L2 cache and all cores share one L3 cache.

- Level 1 (on chip): 32 KiB data cache + 32 KiB instruction cache (8-way associative)
- Level 2 (on chip): 256 KiB cache for data and instructions (8-way associative)
- Level 3 (on chip): 8 MiB cache for data and instructions shared between all cores (16-way associative)

The "Nehalem" processors support a wide variety of x86-instruction-extensions up to SSE4.2. The cores have a clock speed of 2.93 GHz.

2.3.1 Memory

Each processor package (Intel just calls it processor) has its own memory controller and is connected to a local part of the main memory. The processors can access the remote memory via Intel's new interconnect called "Quick Path Interconnect". So these machines are the first Intel

[8] Processor/Thread Binding means explicitly enforcing processes or threads to run on certain processor cores, thus preventing the OS scheduler from moving them around.

[9] The Center for Computing and Communication offers institutes of the RWTH Aachen University to integrate their computers into the HPC-Cluster, where they will be maintained as part of the cluster. The computers will be installed in the center's computer room where cooling and power is provided. Some institutes choose to share compute resources with others, thus being able to use more machines when the demand is high and giving unused compute cycles to others. Further Information can be found at http://www.rz.rwth-aachen.de/go/id/pgo/

Model	Processor type	Sockets/Cores /Threads (total)	Memory Flops/node	Hostname
Sun Fire X4170 (8 nodes)	Intel Xeon X5570 "Nehalem"	2 / 8 / 16 2.93 GHz	36 GiByte 93.76 GFlops	linuxnc001..008
Sun Blade X6275 (192 nodes)	Intel Xeon X5570 "Nehalem"	2 / 8 / 16 2.93 GHz	24 GiByte 93.76 GFlops	linuxnc009..200
Sun Fire X4450 (10 nodes)	Intel Xeon 7460 "Dunnington"	4 / 24 / - 2.66 GHz	128-256 GiByte 255.4 GFlops	linuxdc01..10
Fujitsu-Siemens RX600S4/X (2 nodes)	Intel Xeon E7350 "Tigerton"	4 / 16 / - 2.93 GHz	64 GiByte 187.5 GFlops	linuxhtc01..02
Fujitsu-Siemens RX200S4/X (60 nodes)	Intel Xeon E5450 "Harpertown"	2 / 8 / - 3.0 GHz	16 - 32 GiByte 96 GFlops	linuxhtc03..62 winhtc03..62
IBM eSever LS42 (3 nodes)	AMD Opteron 8356 "Barcelona"	4 / 16 / - 2,3 GHz	32 GiByte 147,2 Gflops	linuxbc01..03

Table 2.3: Node overview

processor-based machines that build a ccNUMA architecture. On ccNUMA computers, processor binding and memory placement are important to reach the whole available performance (see chapter 2.1.1 on page 12 for details).

The machines are equipped with 36 or 24 GiB of DDR3 RAM. The total memory bandwidth is about 37 GB/s.

2.3.2 Network

The Nehalem machines are connected via Gigabit Ethernet and also via quad data rate (QDR) InfiniBand. This QDR InfiniBand achieves an MPI bandwidth of 2.8 GB/s and has a latency of only 2 μs.

2.4 The older Xeon based Machines

2.4.1 The Xeon E5450 "Harpertown" and X7350 "Tigerton" Processors

The Intel Xeon E5450 (Codename "Harpertown") is a 64-bit quad-core processor with two cache levels:

- Level 1 (on chip): Each core has its own 32 KiB data and 32 KiB instruction cache (8-way associative).
- Level 2 (on chip): 2x6 MiB for data and instructions (24-way associative) with full processor clock. Each cache bank is shared between two cores.

These processors support a wide variety of x86 instruction-extensions like MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, Intel 64, EIST, XD-Bit and IVT. The intel64 architecture implements a floating-point stack using 80-bit registers opposed to 64-bit registers in many other processor architectures. Therefore floating point results might slightly differ when using this mode. Some compilers offer a switch to adjust the floating point precision to the IEEE standard. When employing SSE instructions by appropriate compiler options, 64-bit operations are used.

Being a part of the Xeon DP series, the E5450 only supports 2 processors per main board that are connected via a 1333 MHz front-side bus with each other. The cores themselves have

a clock speed of 3 GHz. The X7350 (Codename "Tigerton"), however, is a part of the MP series supporting 4 sockets. The 4 processors, with 4 cores each, are connected via a 1066MHz front-side bus. The cores run at 2.93GHz.

2.4.2 The Xeon X7460 "Dunnington" Processor

The Intel Xeon X7460 (Codename "Dunnington") is the first multi-core (above two) die from Intel. It has six cores on one die and thus per socket, however, the design is similar to the Harpertown processors. Each pair of cores shares a L2 cache and is connected to a common L3 cache.

- Level 1 (on chip): Each core has its own 32 KiB data and 32 KiB instruction 8-way associative cache.
- Level 2 (on chip): 3x3 MiB for data and instructions (24-way associative) running at full processor clock rate. Each cache bank is shared between two cores.
- Level 3 (on chip): All Cores share a 16 MiB L3 Cache for data and instructions (24-way associative) with full processor clock.

The Intel Xeon X7460 supports a wide variety of x86-Instruction-Extensions like MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, Intel 64, EIST, XD-Bit and IVT. The X7460 is a part of the MP Series, supporting 4 sockets. The processor sockets are connected via a 1066 MHz front-side bus, the cores run at 2.66 GHz.

2.4.3 Memory

The cores are pair-wise connected to a L2 cache and each L2 cache with each other via the front-side bus. Access time to the main memory and its bandwidth does *not* depend on the location of the data. In addition, the Dunningtons have a L3 cache that is shared between all cores and is connected to the memory. The machines are equipped with either 16 or 32 GiB (Dunningtons: 128 or 256 GiB) of fully-buffered DDR2 memory providing memory bandwidth of about 7.5 GiB/s (Dunningtons: 10 GiB/s).

2.4.4 Network

Besides Gigabit Ethernet, DDR InfiniBand is used to interconnect these Xeon nodes, except for the nodes with the "Dunnington" CPU.[10] InfiniBand offers a latency of 3.6 μs and a bandwidth up to 1.2 GB/s when using MPI.

2.5 IBM eServer LS42

2.5.1 The Opteron 8356 Processor(Barcelona)

The AMD Opteron 8356 processor (Barcelona) is a 64-bit processor with three cache levels:

- Level 1 (on chip): Each core has 64 KiB for data and 64 KiB for instructions.
- Level 2 (on chip): Each core has 512 KB for data and instructions (16-way associative, 64 Byte cache lines).
- Level 3 (on chip): 2 MB are shared between all cores for data and instructions (32-way associativity, 64 Byte cache lines).

[10] *Note*: Also the nodes with the "Dunnington" CPU are connected via QDR InfiniBand; the network performance of these nodes is known to be somewhat lower.

The Opteron 8356 processor is a quadcore processor and the cores run at 2.3 GHz. They are compatible to the x86 architecture implementing a floating-point stack using 80-bit registers opposed to 64-bit registers used in many other processor architectures. Therefore floating point results will slightly differ. Some compilers offer a switch to adjust the floating point precision to the IEEE standard. When employing SSE instructions by appropriate compiler options, 64-bit operations are used.

2.5.2 Memory

The IBM eServer LS42 machines have 4 processor sockets, each of which is directly attached to 4 GB of local memory. The processors have access to the other processors' memory via the HyperTransport Links, which offer a lower latency than local memory link, but a high bandwidth.

Although there is a high bandwidth rate available, the access to the memory of one processor from another may saturate the memory connection and lead to performance degradation. To avoid these remote accesses of the data, processor binding and data placement are important. See chapter 2.1.1 on page 12 and 3.1.1 on page 17 more for details.

2.5.3 Network

The Opteron nodes are connected via Gigabit Ethernet and DDR InfiniBand, just like the Harpertown nodes.

3 Operating Systems

To accommodate the user's needs we are running two different operating systems on the machines of the HPC-Cluster at the RWTH Aachen University: Linux (see chapter 3.1 on page 17) and Windows (see chapter 3.2 on page 18). The differences between these operating systems are explained in this chapter.

3.1 Linux

Linux is a UNIX-like operating system. We are running the 64-bit version of CentOS Linux, with support for 32-bit binaries, on our systems. CentOS is a clone of RedHat Enterprise Linux.

The CentOS release is displayed by the command:

```
$ cat /etc/issue
```

The Linux kernel version can be printed with the command

```
$ uname -r
```

3.1.1 Processor Binding

Note: The usage of user-defined binding may destroy the performance of other jobs running on the machine. Thus, the usage of user-defined binding is only allowed in batch mode, if cluster nodes are reserved exclusively. Feel free to contact us if you need help with binding issues.

During the runtime of a program, it could happen (and it is most likely) that the scheduler of the operating system decides to move a process or thread from one CPU to another in order to try to improve the load balance among all CPUs of a single node. The higher the system load is, the higher is the probability of processes or threads moving around. In an optimal case this should not happen because, according to our batch job scheduling strategy, the batch job scheduler takes care not to overload the nodes. Nevertheless, operating systems sometimes do not schedule processors in an optimal manner. This may decrease performance considerably because cache contents may be lost and pages may reside on a remote board where they have been first touched. This is particularly disadvantageous on NUMA systems because it is very likely that most of the data accesses will be remote, thus incurring higher latency. *Processor Binding* means that a user explicitly enforces processes or threads to run on certain processor cores, thus preventing the OS scheduler from moving them around.

On Linux you can restrict the set of processors on which the operating system scheduler may run a certain process (in other words, the process is *bound* to those processors). This property is called the *CPU affinity* of a process. The command **taskset** allows you to specify the CPU affinity of a process prior to its launch and also to change the CPU affinity of a running process.

You can get the list of available processors on a system by entering

```
$ cat /proc/cpuinfo
```

The following examples show the usage of **taskset**. We use the more convenient option **-c** to set the affinity with a CPU list (e.g. 0,5,7,9-11) instead of the old-style bitmasks.[11]

[11] The CPUs on which a process is allowed to run are specified with a bitmask in which the lowest order bit corresponds to the first CPU and the highest order bit to the last one.
Running the binary **a.out** on only the first processor:

```
$  taskset 0x00000001 a.out
```

Run on processors 0 and 2:

```
$ $PSRC/pex/320|| taskset 0x00000005 a.out
```

Run on all processors:

```
$  taskset 0xFFFFFFFF a.out
```

If the bitmask is invalid the program will not be executed. An invalid bitmask is e.g. 0x00000010 on a 4-way

`$ $PSRC/pex/321|| taskset -c 0,3 a.out`

You can also retrieve the CPU affinity of an existing task:

`$ taskset -c -p pid`

Or set it for a running program:

`$ taskset -c -p list pid`

Note that the Linux scheduler also supports natural CPU affinity: the scheduler attempts to keep processes on the same CPU as long as this seems beneficial for system performance. Therefore, enforcing a specific CPU affinity is useful only in certain situations.

If using the Intel compilers with OpenMP programs, processor binding of the threads can also be done with the KMP_AFFINITY environment variable (see chapter 6.1.3 on page 55). Similar environment variables for the Oracle compiler are described in section 6.1.4 on page 56 and for the GCC compiler in section 6.1.5 on page 58.

The MPI vendors also offer binding functionality in their MPI implementations; please refer to the documentation.

Furthermore we offer the R_Lib library. It contains portable functions to bind processes and threads (see 9.8 on page 89 for detailed information).

3.2 Windows

The nodes of the Windows part of the HPC-Cluster run Windows Server 2008. All interactive services are disabled on the compute nodes in order to not interfere with compute jobs.

3.3 Addressing Modes

All operating systems on our machines (Linux and Windows) support 64-bit addressing. Programs can be compiled and linked either in 32-bit mode or in 64-bit mode. This affects memory addressing, the usage of 32- or 64-bit pointers, but has no influence on the capacity or precision of floating point numbers (4- or 8-byte real numbers). Programs requiring more than 4 GB of memory have to use the 64-bit addressing mode. You have to specify the addressing mode at compile and link[12] time. The default mode is 32-bit on Windows and 64-bit on Linux. *Note*: *long int* data and pointers in C/C++ programs are stored with 8 bytes when using 64-bit addressing mode, thus being able to hold larger numbers. The example program shown below in listing 1 on page 19 prints out "4" twice in the 32-bit mode:

`$ $CC $FLAGS_ARCH32 $PSRC/pis/addressingModes.c; ./a.out`

and "8" twice in the 64-bit mode:

`$ $CC $FLAGS_ARCH64 $PSRC/pis/addressingModes.c; ./a.out`

machine.

[12] Note the environment variables $FLAGS_ARCH64 and $FLAGS_ARCH32 which are set for compilers by the module system (see chapter 5.2 on page 36).

Listing 1: Show length of pointers and long integer variables

```c
#include <stdio.h>
int main (int argc, char **argv)
{
   int* p;
   long int li;
   printf ("%lu %lu\n",
           (unsigned long int)sizeof(p),
           (unsigned long int)sizeof(li));
   return 0;
}
```

4 The RWTH Environment

4.1 Login to Linux

4.1.1 Command line Login

The secure shell **ssh** is used to log into the Linux systems. Usually **ssh** is installed by default on Linux and Unix systems. Therefore you can log into the cluster from a *local*[13] Unix or Linux machine using the command

`$ ssh -l` *`username`* `cluster.rz.rwth-aachen.de`

Depending on your local configuration it may be necessary to use the **-Y**[14] flag to enable the forwarding of graphical programs. For data transfers use the **scp** command.

A list of frontend nodes you can log into is given in table 1.1 on page 8.

To log into the Linux cluster from a *Windows* machine, you need to have an SSH client installed. Such a client is provided for example by the cygwin (http://www.cygwin.com) environment, which is free to use. Other software is available under different licenses, for example PuTTY (http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html) or SSH Client for Windows (ftp://ftp.cert.dfn.de/pub/tools/net/ssh). To enable the forwarding of graphical programs a *X server* on your Windows computer must run (e.g. the cygwin contains one).

The SSH Client for Windows provides a graphical file manager for copying files to and from the cluster as well (see chapter 4.3.1 on page 23); another tool providing such functionality is WinSCP (http://winscp.net/eng/docs/start).

If you log in over a weak network connection you are welcome to use the **screen** program, which is a full-screen window manager. Even if the connection breaks down, your session will be still alive and you will be able to reconnect to it after you logged in again.[15]

4.1.2 Graphical Login

We are running an NX server on some Linux frontend machines (see table 1.1 on page 8). NX allows you to run remote X11 sessions even across low-bandwidth network connections, as well as reconnecting to running sessions. You can download the NX client from http://www.nomachine.com/download.php

Upon the first time NX is started, the *NX Connection Wizard* will help you to set up the connection. All you need to get started is to enter the session information. By default you will be provided with a KDE desktop from which you can start other programs. If your connection appears to be slow, try out some configuration. Especially enabling *"Configure"* → *"Advanced"* → *"Disable direct draw for screen rendering"* could make your Windows NX client faster. If you are using the KDE graphical desktop environment, you should disable toy features which produce useless updates of the screen. Right-click on the control bar, choose *Configure Panel* (or *Kontrollleiste einrichten* in German), then *Appearance* (*Erscheinungsbild*). In the *General* (*Allgemein*) part, disable both check boxes and save the configuration. Sometimes the environment is broken if using NX, e.g. the LD_LIBRARY_PATH environment variable is not set properly. To repair the environment, use the

```
$ module reload
```

command.

[13] To login from *outside* of the RWTH network you will need either VPN (the recommended way) or a special permit, please contact the support.

[14] older versions of ssh have to use the **-X** option

[15] The **screen** command is known to lose the value of the $LD_LIBRARY_PATH environment variable just after it started. Please use

```
$ module reload
```

command to repair your environment just after you started a new screen.

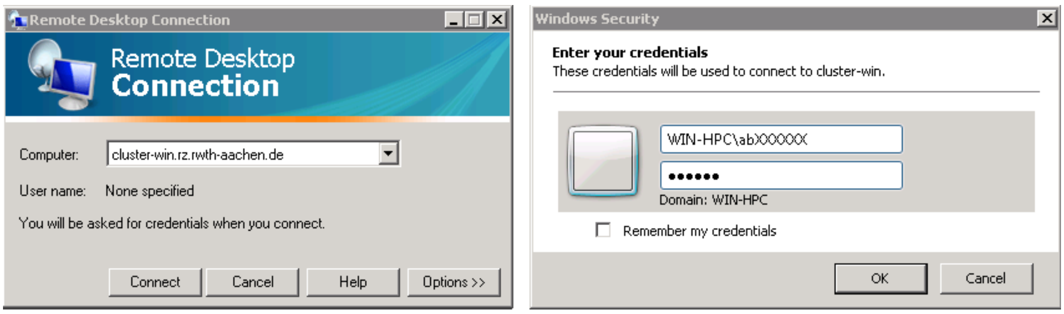
4.2 Login to Windows

We use a load balancing system for the **cluster-win.rz.rwth-aachen.de** frontend that forwards any connection transparently to one of 4 available nodes. Some clients using older versions of the RDP protocol (e.g. all available Linux rdesktop versions and RDP clients of some Windows XP) do not get along with the load balancing system. If you use such a client it might be that you have to repeat the process of entering the domain, your user name and password in a second login screen. That is caused by the transparent redirection of your connection request (you may have noticed that the computer name on the login screen changed). Please enter the domain, your username and password again to login.

The need to enter the user name and password twice is a known problem, but for the time being we cannot come up with a technical solution to this issue.

4.2.1 Remote Desktop Connection

To log into a Windows system from a *Windows environment*, the program *Remote Desktop Connection* is used. You will find it under *"Start"* → *"Programs"* → *"Accessories"* → *"Remote Desktop Connection"* (*„Start"* → *„Programme"* → *„Zubehör"* →*„Remotedesktopverbindung"*). After starting the program, you need to enter the client name (e.g. **cluster-win.rz.rwth-aachen.de**) into the appearing window and then enter your username and password.



Note: Make sure to connect to the *WIN-HPC* domain, because a local login will not work.

You can export local drives or printers to the remote desktop session. Choose **Options** and then the **Local Resources** tab, on the login window of the remote desktop connection, and check the local devices you want to export.

If the *Remote Desktop Connection* program is not installed, it can be downloaded from the Microsoft homepage. *Note*: Administrator privileges are required for installation.

4.2.2 rdesktop, the Linux Client

To log into a Windows system *from a Linux client*, the **rdesktop** program is used. By calling

`$ rdesktop cluster-win.rz.rwth-aachen.de`[16]

you will get a graphical login screen. Frequently used rdesktop options are listed in table 4.4 on page 22. *Note*: Make sure to connect to the *WIN-HPC* domain, because a local login will not work. If called without parameters, **rdesktop** will give information about further options. The following line gives an example of rdesktop usage:

```
$ rdesktop -a 24 -g 90% -r sound:local -r disk:tmp=/tmp -k de -d WIN-HPC
cluster-win.rz.rwth-aachen.de
```

[16] some versions of rdesktop need the **-4** option to work with our windows frontend.

Parameter	Description
-u *user*	Login as *user*
-d *domain*	Use Windows-domain *domain* for authentication
-g *WxH* **-g** *P%*	Desktop geometry *W* (width) x *H* (height), in pixel Use P% of you current screen resolution
-a *depth*	Color depth, *depth*=[8 | 16 | 24], 24 recommended, 8 default
-f	Full-screen mode
-r *device*	Enable specified device or directory redirection
-k *value*	Keyboard layout, e.g. "de" or "us"

Table 4.4: rdesktop options overview

4.2.3 Apple Mac users

Apple Mac users have two alternatives: They can either use **rdesktop** as described above or a native *Remote Desktop Connection Client for Mac*. Please refer http://www.microsoft.com/mac/products/remote-desktop/default.mspx for more information.

4.3 The RWTH User File Management

Every user owns directories on shared file systems (**home** and **work** directories), a scratch directory (**tmp**) and is also welcome to use the **archive** service.

Permanent, long-term data has to be stored in the **home** directory **$HOME=/home/*username*** or on Windows **H:**.Please do not use the home directory for significant amounts of short-lived data because repeated writing and removing creates load on the back-up system. Please use **work** or **tmp** file systems for short-living files. The $HOME data will be backed up in regular intervals. We offer snapshots of the home directory so that older versions of accidentally erased or modified files can be accessed, without requesting a restore from the backup. The snapshots are located in each directory in the **.snapshot/.snapshot/*name*** subdirectory, where the *name* depends on the snapshot interval rule and is *hourly*, *nightly* or *weekly* followed by a number. Zero is the most recent snapshot, higher numbers are older ones. Alternativly, you can access the snapshot of your home directory with the environment variable HOME_SNAPSHOT.

The date of a snapshot is saved in the access time of these directories and can be shown for example with the command
`$ ls -ltru`

The **work** file system is accessible as **$WORK=/work/*username*** or **W:** and is intended for large, medium-term data like intermediate compute results.
Note: Unused data is deleted automatically after 4 weeks in the work file system.

Note: There is no backup of the $WORK file system!

Do not store any non-reproducible or non-recomputable data, like source code or input data, on the work file system! Furthermore, we plan to migrate the work file system from "conventional" file servers to high-performance Lustre file servers (see chapter 4.3.2 on page 23).

Note: Every user has limited space (quota) on **home** and **work** file systems. Use the **quota** command to figure out how much of your space is already used and how much is still available. On the work file system, in addition to the space, also the number of files is limited. Due to the amount of HPC-Cluster users the quota in the home directory is rather small in order to

reduce the total storage requirement. If you need more space or files, please contact us.

Furthermore, the **tmp** directory is available for session-related temporary (scratch) data. Use the **$TMP** environment variable on the Linux (or **%TMP%** on Windows) command line. The directory will be automatically created before and deleted after a terminal session or batch job. Each terminal session and each computer has its own tmp directory, so data sharing is not possible this way! Usually, the tmp file system is mapped onto a local hard disk which provides fast storage. Especially the number of file operations may be many times higher than on network-mounted work and home file systems. However, the size of the tmp file system is rather small and depends on the hardware platform.

Some computers[17] have a network-mounted tmp file system because they do not have sufficient local disk space.

We also offer an **archive** service to store large long-term data, e.g. simulation result files, for future use. A description how to use the archive service can be found at http://www.rz.rwth-aachen.de/li/k/qgy/

4.3.1 Transferring Files to the Cluster

To transfer files to the *Linux* cluster the secure copy command **scp** on Unix or Linux, or the *Secure File Transfer Client* on Windows, can be used. Usually the latter is located in *"Start"* → *"Programs"* → *"SSH Secure Shell"* → *"Secure File Transfer Client"*, if installed. To connect to a system, use the menu *"File"* → *"Quick Connect"*. Enter the host name and user name and select *Connect*. You will get a split window: The left half represents the local computer and the right half the remote system. Files can be exchanged by drag-and-drop. As an alternative to *Secure File Transfer Client*, the *PS-FTP* program can be used, refer to http://www.psftp.de/

If you log into the *Windows* cluster you can export your local drives or directories and access the files as usual. See chapter 4.2.1 on page 21 or 4.2.2 on page 21 for more details. Furthermore you can use the hot keys *ctrl+c* and *ctrl+v* to copy files to and from the remote host.

4.3.2 Lustre Parallel File System

4.3.2.1 Basics

Lustre is a file system designed for high throughput when working with few large files.

Note: When working with many small files (e.g. source code) the Lustre file system may be many times slower than the ordinary network file systems used for $HOME.

To the user it is presented as an ordinary file system, mounted on every node of the cluster. It is planned to use the Lustre file systems for the $WORK mount point in the future.

Note: There is no backup of the Lustre file system!

Programs can perform I/O on the Lustre file system without modification. Nevertheless, if your programs are I/O-intensive, you should consider optimizing them for parallel I/O.

For details on this technology refer to:

- `http://www.lustre.org/`
- `http://www.oracle.com/us/products/servers-storage/storage/storage-software/031855.htm`

4.3.2.2 Mental Model

A Lustre setup consists of one metadata server (MDS) and several object storage servers (OSS).

[17] Currently, only the Sun Blade X6275 computers (see table 2.3 on page 14) have a network-mounted tmp file system (Lustre file system. See 4.3.2 on page 23)

The actual contents of a file are stored in chunks on one or more OSSs, while the MDS keeps track of file attributes (name, size, modification time, permissions, ...) as well as which chunks of the file are stored on which OSS.

Lustre achieves its throughput performance by striping the contents of a file across several OSSs, so I/O performance is not that of a single disk or RAID (hundreds of MB/s), but that of all OSSs combined (up to ~5 GB/s, sequential).

An example: You want to write a 300 MiB file, with a stripe size of 16 MiB (19 chunks), across 7 OSSs. Lustre would pick a list of 7 out of all available OSSs. Then your program would send chunks directly to each OSS like this:

OSS:	1	2	3	4	5	6	7
Chunks:	1	2	3	4	5	6	7
	8	9	10	11	12	13	14
	15	16	17	18	19		

So when your program writes this file, it can use the bandwidth of all requested OSSs, the write operation finishes sooner, and your program has more time left for computing.

4.3.2.3 Optimization

If your MPI application requires large amounts of disk I/O, you should consider optimizing it for parallel file systems. You can of course use the known POSIX APIs (fopen, fwrite, fseek, ...), but MPI as of version 2.0 offers high-level I/O APIs that allow you to describe whole data structures (matrices, records, ...) and I/O operations across several processes. An MPI implementation may choose to use this high-level information to reorder and combine I/O requests across processes to increase performance. The biggest benefit of MPI's parallel I/O APIs is their convenience for the programmer.

Recommended reading:

- "Using MPI-2". Gropp, Lusk, and Thakus. MIT Press.
  Explains in understandable terms the APIs, how they should be used and why.

- "MPI: A Message-Passing Interface Standard", Version 2.0 and later. Message Passing Interface Forum.
  The reference document. Also contains rationales and advice for the user.

4.3.2.4 Tweaks

The **lfs** utility controls the operation of Lustre. You will be interested in **lfs setstripe** since this command can be used to change the stripe size and stripe count. A directory's parameters are used as defaults whenever you create a new file in it. When used on a file name, an empty file is created with the given parameters. You can safely change these parameters; your data will remain intact.

Please do use sensible values though. Stripe sizes should be multiples of 1 MiB, due to characteristics of the underlying storage system. Values larger than 64 MiB have shown almost no throughput benefit in our tests.

4.3.2.5 Caveats

The availability of our Lustre setup is specified as 95 %, which amounts to 1-2 days of expected downtime per month.

Lustre's weak point is its MDS (metadata server); all file operations also touch the MDS, for updates to a file's metadata. Large numbers of concurrent file operations (e.g. a parallel **make** of the Linux kernel) have reliably been resulted in slow down or crash our Lustre setup.

4.4 Defaults of the RWTH User Environment (Lin)

The default login shell is the **Z** (**zsh**) shell. Its prompt is symbolized by the dollar sign. With the special "." dot command a shell script is executed as part of the current process ("sourced"). Thus changes made to the variables from within this script affect the current shell, which is the main purpose of initialization scripts.

```
$ . $PSRC/pex/440
```

For most shells (e.g., bourne shell) you can also use the *source* command:

```
$ source $PSRC/pex/440
```

Environment variables are set with

```
$ export VARIABLE=value
```

This corresponds to the C shell command (the C shell prompt is indicated with a "%" symbol)

```
% setenv VARIABLE value
```

If you prefer to use a different shell, keep in mind to source initialization scripts before you change to your preferred shell or inside of it, otherwise they will run after the shell exits.

```
$ . init_script
$ exec tcsh
```

If you prefer using a different shell (e.g. bash) as default, please append the following lines at THE END of the *.zshrc* file in your home directory:

```
if [[ -o login ]]; then
        bash; exit
fi
```

4.4.1 Z Shell (zsh) Configuration Files

This section describes how to configure the zsh to your needs.

The user configuration files for the zsh are **~/.zshenv** and **~/.zshrc**, which are sourced (in this order) during login. The file **~/.zshenv** is sourced on *every* execution of a zsh. If you want to initialize something e.g. in scripts that use the zsh to execute, put it in ~/.zshenv. Please be aware that this file is sourced during login, too.

Note: Never use a command which calls a zsh in the ~/.zshenv, as this will cause an endless recursion and you will not be able to login anymore.

Note: Do not write to standard output in ~/.zshenv or you will run into problems using **scp**.

In login mode the file **~/.zshrc** is also sourced, therefore ~/.zshrc is suited for interactive zsh configuration like setting aliases or setting the look of the prompt. If you want more information, like the actual path in your prompt, export a format string in the environment variable **PS1**. Example:

```
$ export PS1='%n@%m:%~$'
```

This will look like this:

user@cluster:~/directory$

You can find an example **.zshrc** in $PSRC/psr/zshrc.

You can find further information (in German) about zsh configuration here: http://www.rz.rwth-aachen.de/go/id/owu

4.4.2 The Module Package

The **Module package** provides the dynamic modification of the user's environment. Initialization scripts can be loaded and unloaded to alter or set shell environment variables such as $PATH, to choose for example a specific compiler version or use software packages. The need to load modules will be described in the according software sections in this document.

The advantage of the module system is that environment changes can easily be undone by unloading a module. Furthermore dependencies and conflicts between software packages can be easily controlled. Color-coded warning and error messages will be printed if conflicts are detected.

The **module** command is available for the **zsh**, **ksh** and **tcsh** shells. **csh** users should switch to tcsh because it is backward compatible to csh.
Note: **bash** users have to add the line
. /usr/local_host/etc/bashrc
into ~/**.bashrc** to make the module function available.

The most important options are explained in the following. To get help about the module command you can either read the manual page (man module), or type
`$ module help`
to get the list of available options. To print a list of available initialization scripts, use
`$ module avail`
This list can depend on the platform you are logged in to. The modules are sorted in categories, e.g. CHEMISTRY and DEVELOP. The output may look like the following example, but will usually be much longer.

```
------- /usr/local_rwth/modules/modulefiles/linux/linux64/DEVELOP -------
intel/11.0                          intelmpi/3.1
intel/11.1(default)                 intelmpi/3.2(default)
sunmpi/8.1                          sunmpi/8.2(default)
```

An available module can be loaded with
`$ module load` *modulename*
This will set all necessary environment variables for the use of the respective software. For example, you can either enter the full name like **intelmpi/3.1** or just **intelmpi**, in which case the default **intelmpi/3.2** will be loaded.

A module that has been loaded before but is no longer needed can be removed by
`$ module unload` *modulename*
If you want to use another version of a software (e.g., another compiler), we *strongly recommend*[18] switching between modules:
`$ module switch` *oldmodule newmodule*
This will unload all modules from bottom up to the *oldmodule*, unload the *oldmodule*, load the *newmodule* and then reload all previously unloaded modules. Due to this procedure the order of the loaded modules is not changed and dependencies will be rechecked. Furthermore some modules adjust their environment variables to match previous loaded modules.

You will get a list of loaded modules with
`$ module list`
A short description about the software initialized by a module can be obtained by
`$ module whatis` *modulename*
and a detailed description by
`$ module help` *modulename*
The list of available categories inside of the GLOBAL category can be obtained by
`$ module avail`
To find out in which category a module *modulename* is located try
`$ module apropos` *modulename*
If your environment seems to be insane, e.g. the environment variable $LD_LIBRARY_PATH

[18] The loading of another version by unloading and then loading may lead to a broken environment.

is not set properly, try out

```
$ module reload
```

You can add a directory with your own module files with

```
$ module use path
```

By default, only the DEVELOP software category module is loaded, to keep the available modules clearly arranged. For example, if you want to use a chemistry software you need to load the CHEMISTRY category module. After doing that, the list of available modules is longer and you can now load the software modules from that category.

On Linux the Intel compilers and Oracle's MPI implementation are loaded by default.

Note: If you loaded module files in order to compile a program and subsequently logged out and in again, you probably have to load the same module files before running that program. Otherwise, some necessary libraries may not be found at program startup time. The same situation arises when you build your program and then submit it as a batch job: You may need to put the appropriate module commands in the batch script.

4.5 The RWTH Batch Job Administration

A batch system controls the distribution of *tasks* (also called *batch jobs*) to the available machines and the allocation of other resources which are needed for program execution. It ensures that the machines are not overloaded as this would negatively impact system performance. If the requested resources cannot be allocated at the time the user *submits* the job to the system, the batch job is queued and will be executed as soon as resources become available. Please use the batch system for jobs running longer than 15 minutes or requiring many resources in order to reduce load on the frontend machines.

4.5.1 The Oracle Grid Engine (Lin)

Batch jobs on our Linux systems are handled by the Oracle Grid Engine (OGE).[19] The increasing complexity of computer architecture which is caused by an increasing number of cores per processor chip (CMP=chip-level multiprocessing) and by adding chip-level multithreading (CMT) enforces a change in the way we configure the SGE batch job system.

In the past we used the number of logical processors, which is the number of processors from the operating system's perspective, as the number of job slots on each node. As a consequence, frequently (too) many different user jobs were executed on each single cluster node at a time and mutually affected their performance, which in turn makes it difficult to predict any program's run time. This is particularly harmful if the processes of an MPI job are scattered over multiple cluster nodes and each MPI process has to compete with different other user jobs, because at synchronization points the slowest MPI process will always dominate the overall performance of that MPI job. An obvious solution of this problem would be to reserve the cluster nodes exclusively for single user jobs. But for serial or small parallel jobs running on a cluster with rather fat nodes this would be a waste of resources. Therefore we have been aiming at a compromise between optimizing the cluster's throughput and the single jobs' turnaround time.

We chose a two-step approach:

- For the standard user we want to offer an easy way to use the HPC-Cluster efficiently without running into the pitfalls of the new architectures.
- For advanced users we want to provide the information and the means to further increase their job's efficiency.

[19] Oracle Grid Engine is the new name of well-known Sun Grid Engine (SGE).

The standard user may want to specify (besides other resource parameters like main memory requirements, machine architecture, software licenses etc.)

- the number of MPI processes of an MPI job,
- the number of (OpenMP) threads of a multi-threaded job,
- both of above in the case of a hybrid job,
- and maybe turn on exclusive node usage.

Our Approach: So far the number of job slots per compute node has been equal to the number of logical processors as recognized by the operating system (number of cores on most of x86-based systems without hyper-threading and the number of hardware threads on systems supporting hyper-threading), which occasionally led to overloading the systems and also to mutual performance impacts of multiple jobs running on the same node.
From now on we will therefore specify a number of job slots for each node type which is generally lower than the number of logical processors from the operating system's perspective. A slot contains a set of logical processors. The number of job slots limits the number of units of execution (UEs)[20] which will be executed on a compute node by default. Default is one UE per slot. The knowledgeable user may request an increased number of UEs up to a well-defined limit per compute node of a specific type (see table 4.5 on page 28).

We defined the number of job slots per node for each architecture based on benchmarks. The slot count for each node architecture is chosen such that a single UE will run on a full node with about three-fourths of its performance compared to using the node exclusively. This reflects sharing memory bandwidth, caches and network connections.

CPU / Node type	**Number of Slots**	**UEs per Slot**	**Sockets | Cores | (hardware) Threads per node**
nehalem_typ1	8	2	2 | 8 | 16
barcelona	8	2	4 | 16 | 16
dunnington	12	2	4 | 24 | 24
harpertown	4	2	2 | 8 | 8

Table 4.5: Slot definitions

Dealing with ccNUMA Machines: For shared memory applications running on ccNUMA architectures, the data locality and the thread affinity may be a very important performance impact, as described in chapter 2.1.1.

If one UE needs more memory than locally available, memory will have to be allocated remotely which might disturb the performance of UEs running on this remote NUMA node. However, this effect shouldn't impact the performance of other UEs too much. In case you use binding it is crucial to bind your application to the processors the batch system assigned to your job. Probably processes of other users are running on the other UEs. To avoid a performance drop we prohibit the usage of binding in the batch without consultation of the HPC Group, unless you request the machines for exclusive usage (see table 4.8 on page 31).

Oracle Grid Engine commands: *Note*: At the moment the maximum time limit for a job is 120 hours.[21] If you want to run jobs longer than that and cannot split the computation into

[20] In the following we want to use the term unit of execution (UE) to describe an "entity running on a computer that advanced a program counter" (Tim Mattson, co-author of the book "Patterns for Parallel Programming") which would be a MPI process or a thread of a (MPI) process in this case.
[21] Hosted systems may have different rules.

parts, e.g. via a restart file, please contact us. We also recommend to use a time limit of 24 hours or less, if possible.

Table 4.6 gives a brief overview of the Oracle Grid Engine commands.

jobinfo	Prints information about a batch job.
qsub	Submits a batch job to Oracle Grid Engine. Refer to table 4.7 for parameter details.
qdel	Deletes a batch job with specified *id* from the queue of the Oracle Grid Engine.
qstat	Shows the status of the users batch jobs.
qalter	Modifies a pending or running batch job of Oracle Grid Engine.
qmon	Starts a grapical user's interface for the Oracle Grid Engine.
qselect	Shows the status of jobs and queues.

Table 4.6: Oracle Grid Engine commands

An overview of the current batch job load for the different *clusters*, i.e. sub-clusters of the HPC-Cluster as a whole can be obtained with the RWTH utility **jobinfo**:

`$ jobinfo -c` *`cluster`*

It can also provide you with a list of possible values for *cluster* (see also table 2.3 on page 14):

`$ jobinfo -h`

Job scripts can be submitted to the batch system with the command

`$ qsub [`*`options`*`]` *`scriptfile`*

The most frequently used *parameters of qsub* are shown in table 4.7 on page 30. The most important *resource parameters* (see qsub -l) are listed in table 4.8 on page 31.

Jobs can be deleted with

`$ qdel` *`job_id`*

Status information can be inquired with

`$ qstat [-f | -j` *`job_id`* `| -u` *`user`*`]`

A pending job for example is flagged with *qw*, a running with *r*.

Status information can also be inquired via the graphical user interface (GUI)

`$ qmon`

The attributes of queued jobs can be modified with

`$ qalter` *`options`*

or via the GUI tool **qmon**.

MPI jobs have to be submitted into one of the *parallel environments* presented in table 4.9 on page 31. To specify the architecture of your parallel environment, please use the **mtype** option as described above.

The following simple examples are designed to enable a quick start to submitting a batch job. You can either specify all needed options on the qsub command line, like

```
$ qsub -o $HOME/out.txt -j y -N Test -l h_rt=00:15:00 -l h_vmem=500M -l
threads=2 -l ostype=linux $PSRC/batch/linOpt_Pi
```

or (recommended) write the options in a script file and then invoke qsub with that script as a parameter:

```
$ qsub $PSRC/batch/linOpt_Pi
```

[23] *Note*: If the path name is relative, Oracle Grid Engine expands path either with the current working directory path (if the *-cwd* switch is also specified) or with the home directory path.

[25] *Note*: Exclusive usage of machines is automatically enabled if requesting more than 32 slots. Also note that **-l exclusive** (without *true*) does *not* enable exclusive usage!

[27] *Note*: This option does *not* mean "run N processes on a machine" but only "glue N processes to a bundle and run them on the same node", that is two or more bundles can run on the same machine if there are enough free slots! There is currently no resource to advice the batch system to schedule exactly N processes to a node. You have to adjust other resources (e.g. memory usage or number of threads) to place the desired number of processes across the nodes.

-o [*hostname*:]*path*[23]	redirect standard output to *file*
-e [*hostname*:]*path*[23]	redirect standard error to *file*
-cwd	Execute the job from the current working directory. On start of a batch job, SGE goes to the dir in which the qsub command was executed, instead of (default) user's home dir.
-j y|n	merge error outputs into standard output
-l *resource=value,...*	specification of the necessary resources (see below)
-N *name*	job name
-pe *parallel_environment nproc*	process count for the MPI environment (see below)
-v *variable*[=*value*]	set environment variables
-w v	only check the job parameters assuming an empty cluster, do not submit (this does currently not work in combination with the **-pe** parameter)
-r n	no restart, in case of a system crash (default: restart)
-hold_jid *job_id,...*	start after the termination of the indicated job
-M mail_address	notification mail address
-m b | e | a | s | n	send notification mail at job begin | end | abort | suspend | (default) send no mail
-P project	specify a project. This option is for users associated with a project, e.g. to request hosted systems.

Table 4.7: qsub parameters

Parameter lines in a script have to start with the characters[28] **#$**

It is also possible to mix these two ways. In that case, command line parameters take precedence over embedded script flags.
The following example batch script (Listing 2 on page 30) runs on a Linux machine and writes the output of the program to the **out.txt** file. The job, named Test-Job, may run for 15 minutes and use 500 megabytes of memory. The memory consumption includes all processes belonging to the job, e.g. management processes and shells. You probably have to add 250 megabytes to the memory requirement of your program due to these additionally started processes.

Listing 2: $PSRC/pis/batch_serial.job

```
#$ -o $HOME/out.txt
#$ -j y
#$ -N Test-Job
#$ -l h_rt=00:15:00
#$ -l h_vmem=500M
$PSRC/pex/460  #build the example program a.out
cd 460
a.out
```

The next script (Listing 3 on page 31) is for shared memory parallelized programs, which may e.g. use parallel libraries, automatic parallelization, OpenMP or Pthreads. It also writes the output of the program to the **out.txt** file. The option **-l threads=4** will reserve 4 slots on one node as well as a total[29] 500 megabyte of memory. Furthermore the environment variable OMP_NUM_THREADS is set to the specified processor number, in this case 4.

[28]Note: This character sequence can be accidentally obtained by commenting out a command starting with a variable, for example $MPIEXEC. To avoid errors add a space between # and $.
[29]*Note*: This differs in the MPI case.

-l h_rt=*hh:mm:ss*	Required real time [[hours:]minutes:]seconds Default: 0:15:00 Maximum: 120:00:00 Recommended: less than 24:00:00
-l h_vmem=*xxxX*	Virtual memory per (MPI) processes (sum of all threads of a shared-memory application) specification in bytes, K(bytes), M(bytes) or G(bytes), e.g. vmem=2G; default is : vmem=500M *Note:* To get an idea how much memory your application needs, you can use **memusage *command args***. We advise to start this on the frontend and stop with CRTL+C after some seconds, because in most applications most of the memory is allocated at the begin of the runtime. Now you can round up the virtual memory peak and use as parameter for qsub.
-l mtype=*machine type*	Run only on certain type of machines (see table 4.5 on page 28).
-l *software*=*#_licenses*	The need for software licenses has to be specified. Currently the available licensed packages are abaqus, adams, ansys, cfx5, cfx5_parallel, comsol, fluent, gaussian, hypermesh, lsdyna, marc, marc_64bit, maple, matlab, nastran, powerflow, tascflow, turbomole, starcd, vasp. The number of licenses usually equals 1.
-l threads=*nthr*	In case of shared memory or hybrid parallelization: specification of the number of threads (per each MPI process in case of hybrid parallelization).
-l threadsperslot=*nthr*	Bundle *nthr* threads into one slot.
-l procsperslot=*nproc*	Bundle *nproc* MPI processes into one slot.
-l exclusive=true	Let the jobs run exclusively on the provided machines.[25]
-l hostname=*hostname*	Computer name on which the job has to run. The use of this parameter is not recommended.

Table 4.8: available resources

-pe mpi* *nproc*	*Recommended*: By means of specified resources like memory usage the batch system determines the scheduling pattern by itself.
-pe mpi*1host *nproc*	The MPI job will be started on one single node. All communication will use shared memory.
-pe mpi*2tasks *nproc* **-pe mpi*4tasks** *nproc* **-pe mpi*8tasks** *nproc* **-pe mpi*16tasks** *nproc*	The MPI job will be started with 2, 4, 8 or 16 processes in a process bundle. These bundles will be scheduled on available slots.[27]

Table 4.9: Parallel environments. *Note*: Replace ***nproc*** by the desired number of MPI processes.

Listing 3: $PSRC/pis/batch_omp.job

```
#$ -o $HOME/out.txt
#$ -j y
#$ -N OpenMP-Test-Job
#$ -l h_rt=00:15:00
#$ -l h_vmem=500M
#$ -l threads=4
echo 'Running with ' $OMP_NUM_THREADS 'threads'
$PSRC/pex/460  #build the example program a.out
cd 460; a.out
```

The next example script (Listing 4 on page 32) runs the same program with the same restrictions to memory size and runtime as above. The crucial difference to the last mentioned script is the **-l threadsperslot=2** option. As the name says, this option leads to the start of 2 threads per slot, thus only *two* slots will be reserved for the four threads ordered. The overpopulation of slots with threads may be beneficial, especially on cache-friendly applications, but can also be disadvantageous due to overloading the machine.

Listing 4: $PSRC/pis/batch_omp_adv.job

```
#$ -N OpenMP-Test-Job
#$ -l h_rt=00:15:00
#$ -l h_vmem=500M
#$ -l threads=4
#$ -l threadsperslot=2
echo 'Running with ' $OMP_NUM_THREADS 'threads'
$PSRC/pex/460  #build the example program a.out
cd 460
a.out
```

For MPI programs the parallel environment has to be specified with the **-pe** option. The example in listing 5 on page 32 reserves 5 slots for the 5 MPI processes and 700 megabyte memory for *each* of the processes, resulting in a total of 3.5 gigabytes. The MPI processes may run on one or different nodes. Furthermore the environment variables MPIEXEC and FLAGS_MPI_BATCH are defined by the batch system, according to the loaded MPI module, in order to direct the MPI processes to the reserved machines. *Note*: We strongly recommend to use these environment variables to start MPI programs in batch the way they are stated in the example listings.

Listing 5: $PSRC/pis/batch_mpi.job

```
#$ -N MPI-Test-Job
#$ -l h_rt=00:15:00
#$ -l h_vmem=700M
#$ -pe mpi* 5
$PSRC/pex/461  #build the example program a.out
cd 461
$MPIEXEC $FLAGS_MPI_BATCH a.out
```

The MPI batch job in Listing 6 on page 32 starts 8 MPI processes with 700 megabyte memory for each, resulting in a total of 5.6 GiB. But due to the option **-l procsperslot=2** only *two* slots will be reserved instead of a full slot for every process as in the script in listing 5 on page 32. By using the **-l procsperslot=*nproc*** option a better utilization of hardware may be achieved, but be aware of overloading the machines.

Listing 6: $PSRC/pis/batch_mpi_adv.job

```
#$ -N MPI-Test-Job
#$ -l h_rt=00:15:00
#$ -l h_vmem=700M
#$ -l procsperslot=2
#$ -pe mpi* 8
$PSRC/pex/461  #build the example program a.out
cd 461
$MPIEXEC $FLAGS_MPI_BATCH a.out
```

Hybrid Programs use a combination of MPI and OpenMP, where each MPI process is multi-threaded. The example of a batch job script which starts a hybrid program is given in listing 7 on page 33. It reserves in total 5 (5*1000M) gigabyte of memory and 20 (5*4) slots, which are in 5 groups with 4 slots each. As above, the environment variables MPIEXEC, FLAGS_MPI_BATCH and OMP_NUM_THREADS are set.

Listing 7: $PSRC/pis/batch_hyb.job

```
#$ -N Hybrid-Test-Job
#$ -l h_rt=00:15:00
#$ -l h_vmem=1000M
#$ -pe mpi* 5
#$ -l threads=4
$PSRC/pex/462  #build the example program a.out
cd 462
$MPIEXEC $FLAGS_MPI_BATCH a.out
```

Several example scripts (also for third party (ISV) software packages) can be downloaded at http://www.rz.rwth-aachen.de/go/id/sua (the webpage is in German).

Note: Large MPI processes may run faster if they use machines exclusively. Use the **-l exclusive=true** option to ensure this. If requesting more than 32 slots, the exclusive usage of machines is automatically enabled to enhance the stability and productivity of the cluster. Please note that the exclusive use of machines can increase requested resource usage and thus lead to a higher delay for the job in the queue.

The Oracle Grid Engine allows specifying a big amount of parameters concerning the hardware resources. The requestable flags and values change permanently. For performance reasons qsub does not check if there is a machine available for the requested configuration of resources. If your job does not start, it might be that there are only a few or even no machines that map to your combination of requests. With

`$ qstat -r`

the requested hardware resources of all your jobs are shown (see listing 8 for an example). If you are interested in only one of your jobs, you may combine **-r** with **-j** option:

`$ qstat -r -j` *jobid*

You can print out a list of potentially available machines (in the case of an empty cluster) for a request with

`$ qselect` [**-l** *resource=val*]

If there are no machines shown, you have to lower/change the parameters. In listing **??** for example the value for *r_computeslots* is too high. Note: *qselect* only works with the resource flags, which are printed with *qstat*, not with the defined aliases like *threads*, *threadsperslot* or *procsperslot*.

Listing 8: Example for qstat -r

```
job-ID   prior    name    user         state submit/start at
-------------------------------------------------------------
823515   2.68206 test    ab123456     qw    05/14/2010 13:50:23
  Full jobname:      test
  Hard Resources:    h_fsize=30G (0.000000)
                     h_vmem=500M (0.000000)
                     s_stack=10M (0.000000)
                     h_cpu=15840 (0.000000)
                     r_uesperslot=1 (0.000000)
                     r_ostype=linux (0.000000)
                     r_mem_control=500M (0.000000)
                     r_computeslots=16 (112000.000000)
                     h_rt=900 (0.000000)
                     r_sched=56.25 (-56.250000)
```

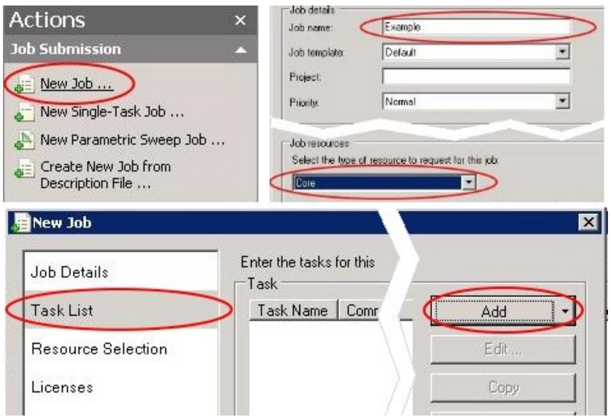
4.5.2 Windows Batch System (Win)

By introducing the *Microsoft HPC Pack*, Microsoft completed its Windows product portfolio for HPC applications. It includes an mpich-2 based MPI-environment (Microsoft MPI) and a batch system with a graphical user interface for job submission and management.

The batch system has two important restrictions: Your program can not accept any user input (if so, it must read it from a file) and nor can it use any elements of the Graphical User Interface (GUI) system. A user's guide is available via *"Start"* → *"All Programs"* → *"Microsoft HPC Pack"* → *"HPC Job Manager Help"*.

To submit a job, you have to start the Cluster Job Manager. For this purpose, choose *"Start"* → *"All Programs"* → *"Microsoft HPC Pack"* → *"HPC Job Manager"* from the Start menu.

To submit a new job, click on "New Job...". The next step is to enter a job name and to select whether you want to use just a core, a socket or a whole node. You should also add a limitation how long your job may run. A job must consist of at least one task, which is the actual execution of a user program or script. Click on "Task List" to add the tasks you want to submit to the cluster. In the new window, enter your commands into the command line, to add a new line, use *Ctrl+Enter*.



It is important that you set a working directory from which the cluster can get the files stated in the command line and where it can put the output and error files.

Remember not to use your windows drives like *H:* as the cluster will only know them if you add a ***net use*** command or if you use the whole network path. The command *net use h: \\cifs\Cluster\Home\<username>* mounts the *$HOME* directory on Linux as the network drive *H:*. Similarly the network drive *W:* can be mounted explicitly: *net use w: \\cifs\Cluster\Work\<username>* *<username>* denotes the 8-digit login name.

In order to access ISV software which is available in *C:\Shared_Software* on the interactive frontend machines, the following network path has to be used in a batch job:
\\cifs\Cluster\Software

You can also specify to which nodes you want to submit your job, but this is not recommended. When you are done, click on *"Submit"* and your job will be queued. With *"Save Job as..."* your configuration will be saved on the disk. It can later be transmitted via *"Actions"* → *"Job Submissions"* → *"Create new Job from Description File"*.

You can check your jobs' status in the *"Active"* window. When it is completed, you will find it either in the table *"Finished"* or, if it failed, in *"Failed"*. If your job does not have a recognizable name, you can identify it with its *"Job ID"*, which you will find out through a Windows balloon after submitting your job. By selecting a job in the job management table further information is available, given you have the necessary rights. A job can be re-configured by right-clicking on it (as long as it still awaits execution) and it can be cancelled as well.

More information about computing on Windows and the windows batch system is available on http://www.rz.rwth-aachen.de/hpc/win web site.

For some software products particular web sites with instructions on how to use them in Windows batch system are available:

MATLAB	http://www.rz.rwth-aachen.de/go/id/sxm/
Abaqus	http://www.rz.rwth-aachen.de/go/id/sxn/
Ansys/Ansys CFX	http://www.rz.rwth-aachen.de/go/id/syh/

5 Programming / Serial Tuning

5.1 Introduction

The basic tool in programming is the compiler, which translates the program source to executable machine code. However, not every compiler is available for the provided operating systems.

On the **Linux** operating system the freely available GNU/GCC[30] compilers are the somewhat "natural" choice. Code generated by these compilers usually performs acceptably on the cluster nodes. From version 4.2 on the GCC compilers offer support for shared memory parallelization with OpenMP. Since version 4 of the GNU compiler suite a FORTRAN 95 compiler – *gfortran* – is available. Code generated by the old *g77* FORTRAN compiler typically does not perform well, so gfortran is recommended.

To achieve the best possible performance on our HPC-Cluster, we recommend using the Intel or Oracle compilers. The Intel compiler family in version 11.1 now provides the default FORTRAN/C/C++/ compilers on our Linux machines. Although the Intel compilers in general generate very efficient code, it can be expected that AMD's processors are not the main focus of the Intel compiler team. As alternatives, the Oracle Studio compilers and PGI compilers are available on Linux, too. Depending on the code, they may offer better performance than the Intel compilers.

The Intel compiler offers interesting features and tools for OpenMP programmers (see chapter 6.1.3 on page 55 and 7.4.2 on page 69). The Oracle compiler offers comparable tools (see chapter 7.4.1 on page 68).

A word of caution: As there is an almost unlimited number of possible combinations of compilers and libraries and also the two addressing modes, 32- and 64-bit, we expect that there will be problems with incompatibilities, especially when mixing C++ compilers.

On **Windows**, the Microsoft Visual Studio environment is installed supporting the Microsoft Visual C++ compiler as well as Intel FORTRAN 95 and C++ compilers.

5.2 General Hints for Compiler and Linker Usage (Lin)

To access non-default compilers you have to load the appropriate module file.[31] You can then access the compilers by their original name, e.g. g++, gcc, gfortran, or via the environment variables $CXX, $CC, or $FC. However, when loading more than one compiler module, you have to be aware that the environment variables point to the *last* compiler loaded.

For convenient switching between compilers, we added environment variables for the most important compiler flags. These variables can be used to write a generic makefile that compiles with any loadable compiler. The offered variables are listed below. Values for different compilers are listed in tables 5.10 on page 37 and 6.14 on page 54.

- $FC, $CC, $CXX – a variable containing the appropriate compiler name.
- $FLAGS_DEBUG – enables debug information.
- $FLAGS_FAST – includes the options which usually offer good performance. For many compilers this will be the **-fast** option. But beware of possible incompatibility of binaries, especially with older hardware.
- $FLAGS_FAST_NO_FPOPT – equally to FAST, but disallows any floating point optimizations which will have an impact on rounding errors.
- $FLAGS_ARCH32, $FLAGS_ARCH64 – builds 32 or 64 bit executables or libraries.

[30] GCC, the GNU Compiler Collection: http://gcc.gnu.org/
[31] see chapter 4.4.2 on page 25.

- $FLAGS_AUTOPAR – enable auto-parallelization, if supported by the compiler.
- $FLAGS_OPENMP – enables OpenMP support, if supported by the compiler.
- $FLAGS_RPATH – contains a set of directories (addicted to loaded modules) to add to the runtime library search path of the binary, with a compiler-specific command (according to the last loaded compiler) to pass these paths to the linker.[32]

In order to be able to mix different compilers all these variables (except $FLAGS_RPATH) also exist with the compiler's name in the variable name, such as $GCC_CXX or $FLAGS_GCC_FAST.
Example:
$ $PSRC/pex/520|| `$CXX $FLAGS_FAST $FLAGS_ARCH64 $FLAGS_OPENMP $PSRC/cpop/pi.cpp`

The makefiles of the example programs also use these variables (see chapter 1.3 on page 9 for further advice on using these examples).

Flag ↓ Compiler →	Oracle	Intel	GCC
$FLAGS_DEBUG	-g | -g0	-g	-g
$FLAGS_FAST	-fast	-O3 -axSSE4.2,SSSE3,SSE2 -fp-model fast=2	-O3 -ffast-math
$FLAGS_FAST_NO_FPOPT	-fast -fsimple=0	-O3 -axSSE4.2,SSSE3,SSE2 -fp-model precise	-O3
$FLAGS_ARCH32|64	-m32 | -m64	-m32 | -m64	-m32 | -m64

Table 5.10: Compiler options overview

In general we strongly recommend using the same flags for both compiling and linking. Otherwise the program may not run correctly or linking may fail.

The order of the command line options while compiling and linking does matter.

The rightmost compiler option, in the command line, takes precedence over the ones on the left, e.g. **cc ... -O3 -O2**. In this example the optimization flag O3 is overwritten by O2. Special care has to be taken if macros like **-fast** are used because they may overwrite other options unintentionally. Therefore it is advisable to enter macro options at the beginning of the command line.

If you get unresolved symbols while linking, this may be caused by a wrong order of libraries. If a library xxx uses symbols from the library yyy, the library yyy has to be right of xxx in the command line, e.g. **ld ... -lxxx -lyyy**.

The search path for header files is extended with the **-I*directory*** option and the library search path with the **-L*directory*** option.

The environment variable LD_LIBRARY_PATH specifies the search path where the program loader looks for shared libraries. Some compile time linkers (e.g., the Oracle linker) also use this variable while linking, but the GNU linker does not.

Consider the static linking of libraries. This will generate a larger executable, which is however a lot more portable. Especially on Linux the static linking of libraries may be a good idea since every distribution has slightly different library versions which may not be compatible with each other.

5.3 Tuning Hints

There are some excellent books covering tuning application topics have been published:

[32] If linked with this option, the binary "knows" at runtime where its libraries are located and is thus independent of which modules are loaded at the runtime.

- Rajat Garg and Ilya Sharapov: Techniques for Optimizing Applications: High Performance Computing, ISBN:0-13-093476-3, published by Prentice-Hall PTR/Sun Press.
- K. Dowd, C. Severance: High Performance Computing, O'Reilly, 2nd edition, 1998.
- S. Goedecker, A. Hoisie: Performance Optimization of Numerically Intensive Codes, ISBN:0-89871-484-2, published by Society for Industrial Mathematics.

Contiguous memory access is crucial for reducing cache and TLB misses. This has a direct impact on the addressing of multidimensional fields or structures. FORTRAN arrays should therefore be accessed by varying the leftmost indices most quickly and C and C++ arrays with rightmost indices. When using structures, all structure components should be processed in quick succession. This can frequently be achieved with *loop interchange*.

The limited memory bandwidth of processors can be a severe bottleneck for scientific applications. With *prefetching* data can be loaded prior to the usage. This will help reducing the gap between the processor speed and the time it takes to fetch data from memory. Such a prefetch mechanism can be supported automatically by hardware and software but also by explicitly adding prefetch directives (FORTRAN) or function calls in C and C++.

The re-use of cache contents is very important in order to reduce the number of memory accesses. If possible, blocked algorithms should be used, perhaps from one of the optimized numerical libraries described in chapter 9 on page 85.

Cache behavior of programs can be improved frequently by *loop fission* (=*loop splitting*), *loop fusion* (=*loop collapsing*, *loop unrolling*, *loop blocking*, *strip mining*), and combinations of these methods. Conflicts caused by the mapping of storage addresses to the same cache addresses (*false sharing*) can be eased by the creation of buffer areas (*padding*).

The compiler optimization can be improved by integrating frequently called small subroutines into the calling subroutines (*inlining*). This will not only eliminate the cost of a function call, but also give the compiler more visibility into the nature of the operations performed, thereby increasing the chances of generating more efficient code.

Consider the following general program tuning hints:

- Turn on high optimization while compiling. The use of $FLAGS_FAST options may be a good starting point. However keep in mind that optimization may change rounding errors of floating point calculations. You may want to use the variables supplied by the compiler modules. An optimized program runs typically 3 to 10 times faster than the non-optimized one.
- Try another compiler. The ability of different compilers to generate efficient executables varies. The runtime differences are often between 10% and 30%.
- Write efficient code that can be optimized by the compiler. We offer videos of several talks that are a good introduction into this topic, see
  http://www1.rz.rwth-aachen.de/computing/events/material.php

  Furthermore we offer materials from tuning workshops on this website: http://www.rz.rwth-aachen.de/go/id/nwy
- Try to perform as little input and output as possible and bundle it into larger chunks.
- Try to allocate big chunks of memory instead of many small pieces, e.g. use arrays instead of linked lists, if possible.
- Access memory continuously in order to reduce cache and TLB misses. This especially affects multi-dimensional arrays and structures. In particular, note the difference between FORTRAN and C/C++ in the arrangement of arrays! Tools like Acumem (chapter 8.4 on page 76) or Intel VTune/PTU (chapter 8.2.1 on page 74) may help to identify problems easily.

- Use a profiling tool (see chapter 8 on page 71), like the Oracle (Sun) Collector and Analyzer, Intel VTune/PTU or gprof to find the computationally intensive or time-consuming parts of your program, because these are the parts where you want to start optimization.
- Use optimized libraries, e.g. the Intel MKL, the Oracle (Sun) Performance Library or the ACML library (see chapter 9 on page 85).
- Consider parallelization to reduce the runtime of your program.

5.4 Endianness

In contrast to e.g. the UltraSPARC architecture, the x86 AMD and Intel processors store the least significant bytes of a native data type first (**little endian**). Therefore care has to be taken if binary data has to be exchanged between machines using **big endian** – like the UltraSPARC-based machines – and the x86-based machines. Typically, FORTRAN compilers offer options or runtime parameters to write and read files in different byte ordering.

For other programming languages than FORTRAN the programmer has to take care of swapping the bytes when reading binary files. Below is a C++ example to convert from big to little endian or vice versa. This example can easily be adapted for C; however, one has to write a function for each data type since C does not know templates.
Note: This only works for basic types, like integer or double, and not for lists or arrays. In case of the latters, every element has to be swapped.

Listing 9: $PSRC/542

```
template <typename T> T swapEndian( T x){
    union{ T x; unsigned char b[sizeof(T)];} dat1, dat2;

    dat1.x = x;
    for (int i = 0; i < sizeof(T); ++i)
    {
        dat2.b[i] = dat1.b[sizeof(T)-1-i];
    }
    return dat2.x;
}
```

5.5 Intel Compilers (Lin / Win)

On Linux, a version of the Intel FORTRAN/C/C++ compilers is loaded into your environment per default. They may be invoked via the environment variables $CC, $CXX, $FC or directly by the commands **icc** | **icpc** | **ifort** on Linux and **icl** | **ifort** on Windows. The corresponding manual pages are available for further information. An overview of all the available compiler options may be obtained with the flag **-help**.

You can check the version which you are currently using with the **-V** option. Please use the `module` command to switch to a different compiler version. You can get a list of all the available versions with *module avail intel*. In general, we recommend using the latest available compiler version to benefit from performance improvements and bug fixes.

On Windows, the Intel Compilers can be used either in the Visual Studio environment or on the Cygwin command line.

5.5.1 Frequently Used Compiler Options

Compute intensive programs should be compiled and linked (!) with the optimization options which are contained in the environment variable $FLAGS_FAST. For the Intel compiler,

$FLAGS_FAST currently evaluates to

```
$ echo $FLAGS_FAST
-O3 -axSSE4.2,SSSE3,SSE2 -fp-model fast=2
```

These flags have the following meaning:

- **-O3**: This option turns on aggressive, general compiler optimization techniques. Compared to the less aggressive variants **-O2** and **-O1**, this option may result in longer compilation times, but generally faster execution. It is especially recommended for code that processes large amounts of data and does a lot of floating-point calculations.
- **-axSSE4.2,SSSE3,SSE2**: This option turns on the automatic vectorizer[33] of the compiler and enables code generation for processors which employ the vector operations contained in the SSE4.2, SSSE3, and SSE2 instruction set extensions. Compared to the similar option **-xSSE4.2,SSSE3,SSE2**, this variant also generates machine code which does not use the vector instruction set extensions so that the executable can also be run on processors without these enhancements. This is reasonable on our HPC-Cluster, because not all of our machines support the same instruction set extensions.
- **-fp-model fast=2**: This option enables aggressive optimizations of floating-point calculations for execution speed, even those which might decrease accuracy.

Other options which might be of particular interest to you are:

- **-openmp**: Turns on OpenMP support. Please refer to Section 6.1 on page 53 for information about OpenMP parallelization.
- **-parallel**: Turns on auto-parallelization. Please refer to Section 6.1 on page 53 for information about auto-parallelizing serial code.
- **-vec-report**: Turns on feedback messages from the vectorizer. If you instruct the compiler to vectorize your code (e.g. by using **-axSSE4.2,SSSE3,SSE2**) you can make it print out information about which loops have successfully been vectorized with this flag. Usually, exploiting vector hardware to its fullest requires some code re-structuring which may be guided by proper compiler feedback. To get the most extensive feedback from the vectorizer, please use the option **-vec-report3**. As the compiler output may become a bit overwhelming in this case, you can instruct the compiler to only tell about failed attempts to vectorize (and the reasons for the failure) by using **-vec-report5**.
- **-convert big_endian**: Read or write big-endian binary data in FORTRAN programs.

Table 5.11 on page 41 provides a concise overview of the Intel compiler options.

5.5.2 Tuning Tips

5.5.2.1 The Optimization Report To fully exploit the capabilities of an optimizing compiler it is usually necessary to re-structure the program code. The Intel Compiler can assist you in this process via various reporting functions. Besides the vectorization report (cf. Section 5.7.1 on page 47) and the parallelization report (cf. Section 6.1.3 on page 55), a general optimization report can be requested via the command line option **-opt-report**. You can control the level of detail in this report; e.g. **-opt-report 3** provides the maximum amount of optimization messages.

[33] Intel says, for the Intel Compiler, vectorization is the unrolling of a loop combined with the generation of packed SIMD instructions.

[37] For this option the syntax -Ob*N* is still available on Linux but is deprecated.

[38] The letter values for *processor* are deprecated now.

[39] Objects compiled with -ipo are not portable, so do not use for libraries.

Linux	Windows	Description
-c	/c	compile, but do not link
-o *filename*	/Fo *filename*	specify output file name
-O0	/Od	no optimization (useful for debugging)
-O1	/O1	some speed optimization
-O2	/O2	(default) speed optimization, the generated code can be significantly larger
-O3	/O3	highest optimization, may result in longer compilation times
-fast	/fast	a simple, but less portable way to get good performance. The -fast option turns on -O3, -ipo, -static and -no-prec-div. *Note*: A processor with SSE3 extensions is required, this option will not work on older Opterons. *Note*: -no-prec-div enables optimizations that give slightly less precise results than full IEEE division.
-inline-level=N[37]	/ObN	N = 0: disable inlining(default if -O0 specified) N = 1: enable inlining(default) N = 2: automatic inlining
-xC	/QxC	generate code optimized for *processor*[38]extensions. The code will only run on this platform. *SSE4.2*: newer SSE4, e.g.in Nehalem CPU's *SSE4.1*: older SSE4 extensions *SSSE3*: SSE3 for e.g. Woodcrest or Clowertown *SSE2*: SSE2 for e.g. older Opterons
-axC_1,C_2,..	/QaxC_1,C_2,..	like -x, but you can optimize for several platforms, and baseline code path is also generated
-vec-reportX	/Qvec-reportX	emits level X diagnostic information from the vectorizer; if X is left out, level 1 is assumed
-ip	/Qip	enables additional interprocedural optimizations for single-file compilation
-ipo	/Qipo	enables interprocedural optimization between files Functions from different files may be inlined[39]
-openmp	/Qopenmp	enables generation of parallel code based on OpenMP directives
-openmp-stubs	/Qopenmp-stubs	compiles OpenMP programs in sequential mode; the OpenMP directives are ignored and a sequential version of the OpenMP library is linked
-parallel	/Qparallel	generates multi-threaded code for-loops that can be safely executed in parallel (auto-parallelization)
-par-reportX	/Qpar-reportX	emit level X diagnostic information from the auto-parallelizer; if X is left out, level 1 is assumed
-tcheck -tprofile	/Qtcheck /Qtprofile	enables analysis of threaded applications with Intel(R) Thread Checker or Profiler
-g	/Zi	produces symbolic debug information in object file
	/stack:*size*	set the default stack size in byte
-Xlinker *val*	/link *val*	passes *val* directly to the linker for processing

Table 5.11: Intel Compiler Options

The amount of feedback generated by this compiler option can easily get overwhelming. Therefore, you can put the report into a file (**-opt-report-file**) or restrict the output to a certain compiler phase (**-opt-report-phase**) or source code routine (**-opt-report-routine**).

5.5.2.2 Interprocedural Optimization Traditionally, optimization techniques have been limited to single routines because these are the units of compilation in FORTRAN. With interprocedural optimization, the compiler extends the scope of applied optimizations to multiple routines, potentially to the program as a whole. With the flag **-ip**, interprocedural optimization can be turned on for a single source file, i.e. the possible optimizations cover all routines in that file. When using the **-O2** or **-O3** flags, some single-file interprocedural optimizations are already included.

If you use **-ipo** instead of **-ip**, you turn on multi-file interprocedural optimization. In this case, the compiler does not produce the usual object files, but mock object files which include information used for the optimization.

The **-ipo** option may considerably increase the link time. Also, we often see compiler bugs with this option. The performance gain when using **-ipo** is usually moderate, but may be dramatic in object-oriented programs. Do not use **-ipo** for producing libraries because object files are not portable if -ipo is on.

5.5.2.3 Profile-Guided Optimization When trying to optimize a program during compile/link time, a compiler can only use information contained in the source code itself or otherwise supplied to it by the developer. Such information is called "static" because it is passed to the compiler before the program has been built and hence does not change during runtime of the program. With Profile-Guided Optimization, the compiler can additionally gather information during program runs (dynamic information).

You can instrument your code for Profile-Guided Optimization with the **-prof-gen** flag. When the executable is run, a profile data file with the .dyn suffix is produced. If you now compile the source code with the **-prof-use** flag, all the data files are used to build an optimized executable.

5.5.3 Debugging

The Intel compiler offers several options to help you find problems with your code:

- **-g**: Puts debugging information into the object code. This option is necessary if you want to debug the executable with a debugger at the source code level (cf. Chapter 7 on page 65). Equivalent options are: **-debug**, **-debug full**, and **-debug all**.
- **-warn**: Turns on all warning messages of the compiler.
- **-O0**: Disables any optimization. This option accelerate the compilations during the development/debugging stages.
- **-gen-interfaces**: (FORTRAN only) Creates an interface block (a binary .mod file and the corresponding source file) for each subroutine and function.
- **-check**: (FORTRAN only) Turns on runtime checks (cf. Chapter 7.2 on page 66).

5.6 Oracle Compilers (Lin)

On the Linux-based nodes, the Oracle[40] Studio 12.1 development tools are now in production mode and available after loading the appropriate module with the *module* command (see section 4.4.2 on page 25). They include the FORTRAN 95, C and C++ compilers. If necessary you

[40] formerly Sun

can use other versions of the compilers by modification of the search path through loading the appropriate module with the *module* command (see section 4.4.2 on page 25).

```
$ module switch studio studio/12
```

Accordingly you can use preproduction releases of the compiler, if they are installed. You can obtain the list of all available versions by *module avail studio*.

We recommend that you always recompile your code with the latest production version of the used compiler due to performance reasons and bug fixes. Check the compiler version that you are currently using with the compiler option **-V**.

The compilers are invoked with the commands

```
$ cc, c89, c99, f90, f95, CC
```

and since Studio 12 additional Oracle-specific names are available

```
$ suncc, sunc89, sunc99, sunf90, sunf95, sunCC
```

You can get an overview of the available compiler flags with the option **-flags**. We strongly recommended using the same flags for both compiling and linking.

From the Sun Studio 7 Compiler Collection release on, a separate FORTRAN 77 compiler is no longer available. **f77** is a wrapper script used to pass the necessary compatibility options, like **-f77**, to the f95 compiler. This option has several suboptions. Using this option without any explicit suboption list expands to **-ftrap=%none -f77=%all**, which enables all compatibility features and also mimics FORTRAN 77's behavior regarding arithmetic exception trapping. We recommend adding **-f77 -ftrap=common** in order to revert to f95 settings for error trapping, which is considered to be safer. When linking to old f77 object binaries, you may want to add the option **-xlang=f77** at the link step. Detailed information about compatibility issues between FORTRAN 77 and FORTRAN 95 can be found in **http://docs.sun.com/source/816-2457/5_f77.html**

For information about shared memory parallelization see chapter 6.1.4 on page 56.

5.6.1 Frequently Used Compiler Options

Compute-intensive programs should be compiled and linked (!) with the optimization options which are contained in the environment variable $FLAGS_FAST.[41]

Since the Studio compiler may produce 64bit binaries as well as 32bit binaries and the default behavior is changing across compiler versions and platforms, we recommend setting the bit width explicitly by using the $FLAGS_ARCH64 or $FLAGS_ARCH32 environment variables.

The often-used option **-fast** is a macro expanding to several individual options that are meant to give the best performance with one single compile and link option. Note, however, that the expansion of the **-fast** option might be different across the various compilers, compiler releases, or compilation platforms. To see to which options a macro expands use the **-v** or **-#** options. On our Nehalem machines this looks like:

```
$ CC -v -fast $PSRC/cpsp/pi.cpp -c
### command line files and options (expanded):
### -v -xO5 -xarch=sse4_2 -xcache=32/64/8:256/64/8:8192/64/16 -xchip=nehalem
-xdepend=yes -fsimple=2 -fns=yes -ftrap=%none -xlibmil -xlibmopt
-xbuiltin=%all -D__MATHERR_ERRNO_DONTCARE -nofstore -xregs=frameptr -Qoption
CC -iropt -Qoption CC -xcallee64 /rwthfs/rz/SW/HPC/examples/cpsp/pi.cpp -c
-Qoption ube -xcallee=yes
```

The compilers on x86 do not use automatic prefetching by default, nor does the C++ compiler on Sparc. Turning prefetching on with the **-xprefetch** option might offer better performance. Some options you might want to read up on are: **-xalias_level**, **-xvector**,

[41] Currently, on Linux the environment variables $FLAGS_FAST and $FLAGS_FAST_NO_FPOPT contain flags which optimize for the Intel Nehalem CPU's. On older chips, there may be errors with such optimized binaries due to lack of SSE4 units. Please read the compiler man page carefully to find out the best optimization flag for the chips you want your application to run on.

-xspfconst and **-xprefetch**. These options only offer better performance in some cases and are therefore not included in the **-fast** macro.

Note: High optimization can have an influence on floating point results due to different **rounding errors**. To keep the order of the arithmetic operations, additional options **-fsimple=0** or **-xnolibmopt** can be added, which, however, may reduce the execution speed; see the $FLAGS_FAST_NO_FPOPT environment variable.

On the x86 nodes the **rounding precision** mode can be modified when compiling a program with the option[42] **-fprecision=single | double | extended**. The following code snippet demonstrates the effect:

Listing 10: $CC $FLAGS_ARCH32 $PSRC/pis/precision.c; a.out

```
#include <stdio.h>
int main (int argc, char **argv)
{
    double f = 1.0, h = 1.0;
    int i;
    for (i = 0; i < 100; i++)
    {
       h = h / 2;
       if (f + h == f) break;
    }
    printf ("f: %e h: %e mantissa bits: %d\n", f, h, i);
    return 0;
}
```

Results	**x86 32bit, no SSE2**	**other**
`1.000000e+00 5.960464e-08 23`	-fprecision=single	n.a.
`1.000000e+00 1.110223e-16 52`	-fprecision=double	(*default*)
`1.000000e+00 5.421011e-20 63`	-fprecision=extended (*default*)	n.a.

Table 5.12: Results of different rounding modes

The results are collected in table 5.12 on page 44. The mantissa of the floating point numbers will be set to 23, 52 or 63 bits respectively. If the compiling is done on UltraSPARC nodes or on x86 nodes in 64bit or in 32bit with the usage of SSE2 instructions, the option **-fprecision** is ignored and the mantissa is always set to 52 bits.

The Studio FORTRAN compiler supports unformatted file sharing between big-endian and little-endian platforms (see chapter 5.4 on page 39) with the option **-xfilebyteorder=*endianmaxalign*:*spec*** where *endian* can be one of **little**, **big** or **native**, *maxalign* can be **1**, **2**, **4**, **8** or **16** specifying the maximum byte alignment for the target platform, and *spec* is a filename, a FORTRAN IO unit number, or **%all** for all files. The default is **-xfilebyteorder=native:%all**, which differs depending on the compiler options and platform. The different defaults are listed in table 5.13 on page 44.

32 bit addressing	**64 bit addressing**	**architecture**
`little4:%all`	`little16:%all`	x86
`big8:%all`	`big16:%all`	UltraSPARC

Table 5.13: Endianness options

The *default data type mappings* of the FORTRAN compiler can be adjusted with the **-xtypemap** option. The usual setting is **-xtypemap=real:32,double:64,integer:32**.

[42] *Note*: this works only if the program is compiled in 32bit and does not use SSE2 instructions. The man page of Oracle compiler does not say this clear.

The *REAL* type for example can be mapped to 8 bytes with **-xtypemap=real:64,double:64,integer:32**.

The option **-g** writes *debugging information* into the generated code. This is also useful for runtime analysis with the Oracle (Sun) Performance Analyzer that can use the debugging information to attribute time spent to particular lines of the source code. Use of **-g** does not substantially impact optimizations performed by the Oracle compilers. On the other hand, the correspondence between the binary program and the source code is weakened by optimization, making debugging more difficult. To use the Performance Analyzer with a *C++* program, you can use the option **-g0** in order not to prevent the compiler of inlining. Otherwise performance might drop significantly.

5.6.2 Tuning Tips

The option **-xunroll=n** can be used to advise the compiler to unroll loops.

Conflicts caused by the mapping of storage addresses to cache addresses can be eased by the creation of buffer areas (*padding*) (see compiler option **-pad**).

With the option **-dalign** the memory access on 64-bit data can be accelerated. This alignment permits the compiler to use single 64-bit load and store instructions. Otherwise, the program has to use two memory access instructions. If **-dalign** is used, every object file has to be compiled with this option.

With this option, the compiler will assume that double precision data has been aligned on an 8 byte boundary. If the application violates this rule, the runtime behavior is undetermined, but typically the program will crash. On well-behaved programs this should not be an issue, but care should be taken for those applications that perform their own memory management, switching the interpretation of a chunk of memory while the program executes. A classical example can be found in some (older) Fortran programs in which variables of a COMMON block are not typed consistently.

The following code will break, i.e. values other than 1 are printed, when compiled with the option **-dalign**:

Listing 11: f90 -dalign $PSRC/pis/badDalignFortran.f90; a.out

```
Program verybad
    call sub1
    call sub2
end Program
subroutine sub1
   integer a, b, c, d
   common / very_bad / a, b, c, d
   d=1
end subroutine sub1
subroutine sub2
   integer a, d
   real*8 x
   common / very_bad / a, x, d
   print *, d
end subroutine sub2
```

Note: The option **-dalign** is actually *required* for Fortran MPI programs and for programs linked to other libraries like the Oracle (Sun) Performance Library and the NAG libraries.

Inlining of routines from the same source file:
-xinline=*routine1,routine2,...*
However, please remember that in this case automatic inlining is disabled. It can be restored through the %auto option. We therefore recommend the following:

-xinline=%auto,*routine_list*.

With optimization level -xO4 and above, this is automatically attempted for functions / subroutines within the same source file. If you want the compiler to perform inlining across various source files at linking time, the option **-xipo** can be used. This is a compile and link option to activate interprocedural optimization in the compiler. Since the 7.0 release, **-xipo=2** is also supported. This adds memory-related optimizations to the interprocedural analysis.

In C and C++ programs, the use of pointers frequently limits the compiler's optimization capability. Through compiler options **-xrestrict** and **-xalias_level=...** it is possible to pass on additional information to the C-compiler. With the directive

```
#pragma pipeloop(0)
```

in front of a *for* loop it can be indicated to the C-compiler that there is no data dependency present in the loop. In FORTRAN the syntax is

```
!$PRAGMA PIPELOOP=0
```

Attention: These options (**-xrestrict** and **-xalias_level**) and the *pragma* are based on certain assumptions. When using these mechanisms incorrectly, the behavior of the program becomes undefined. Please study the documentation carefully before using these options or directives.

Program kernels with numerous branches can be further optimized with the profile feedback method. This two-step method starts with compilation using this option added to the regular optimization options **-xprofile=collect:*a.out***. Then the program should be run for one or more data sets. During these runs, runtime characteristics will be gathered. Due to the instrumentation inserted by the compiler, the program will most likely run longer. The second phase consists of recompilation using the runtime statistics **-xprofile=use:*a.out***. This produces a better optimized executable, but keep in mind that this is only beneficial for specific scenarios.

When using the **-g** option and optimization, the Oracle compilers introduce **comments** about loop optimizations into the object files. These comments can be printed with the command

$ $PSRC/pex/541|| `er_src serial_pi.o`

A comment like *Loop below pipelined with steady-state cycle count...* indicates that software pipelining has been applied, which in general results in better performance. A person knowledgeable of the chip architecture will be able to judge by the additional informationwhether further optimizations are possible.

With a combination of `er_src` and grep, successful subroutine inlining can also be easily verified

$ $PSRC/pex/541|| `er_src *.o |grep inline`

5.6.3 Interval Arithmetic (Lin)

The Oracle FORTRAN and C++ compilers support interval arithmetic. In FORTRAN this is implemented by means of an intrinsic INTERVAL data type, whereas C++ uses a special class library. The use of interval arithmetic requires the use of appropriate numerical algorithms. For more information, see http://docs.sun.com/app/docs/doc/819-3695 web pages.

5.7 GNU Compilers (Lin)

On Linux, a version of the GNU compilers is always available because it is shipped with the operating system, although this system-default version may be heavily outdated. Please use the **module**[43] command to switch to a non-default GNU compiler version.

The GNU FORTRAN/C/C++ compilers can be accessed via the environment variables $CC, $CXX, $FC (if the gcc module is the last loaded module) or directly by the commands **gcc | g++ | g77 | gfortran**.

The corresponding manual pages are available for further information. The FORTRAN 77 compiler understands some FORTRAN 90 enhancements, when called with the parameters

[43] see chapter 4.4.2 on page 25

-ff90 -ffree-form. Sometimes the option **-fno-second-underscore** helps in linking. The FORTRAN 95 Compiler **gfortran** is available since version 4.

5.7.1 Frequently Used Compiler Options

Compute-intensive programs should be compiled and linked (!) with the optimization options which are contained in the environment variable $FLAGS_FAST. For the GNU compiler 4.4, $FLAGS_FAST currently evaluates to

```
$ echo $FLAGS_FAST
-O3 -ffast-math -mtune=native
```

These flags have the following meaning:

- **-O3**: The **-Ox** options control the number and intensity of optimization techniques the compiler tries to apply to the code. Each of these techniques has individual flags to turn it on, the **-Ox** flags are just summary options. This means that **-O** (which is equal to **-O1**) turns some optimizations on, **-O2** a few more, and **-O3** even more than **-O2**.
- **-ffast-math**: With this flag the compiler tries to improve the performance of floating point calculations while relaxing some correctness rules. **-ffast-math** is a summary option for several flags concerning floating point arithmetic.
- **-mtune=native**: Makes the compiler tune the code for the machine on which it is running. You can supply this option with a specific target processor; please consult the GNU compiler manual for a list of available CPU types. If you use **-march** instead of **-mtune**, the generated code might not run on all cluster nodes anymore, because the compiler is free to use certain parts of the instruction set which are not available on all processors. Hence, **-mtune** is the less aggressive option and you might consider switching to **-march** if you know what you are doing.

Other options which might be of particular interest to you are:

- **-fopenmp**: Enables OpenMP support (GCC 4.2 and newer versions). Please refer to Section 6.1 on page 53 for information about OpenMP parallelization.
- **-ftree-parallelize-loops=N**: Turns on auto-parallelization and generates an executable with N parallel threads (GCC 4.3 and newer versions). Please refer to Section 6.1 on page 53 for information about auto-parallelizing serial code.

5.7.2 Debugging

The GNU compiler offers several options to help you find problems with your code:

- **-g**: Puts debugging information into the object code. This option is necessary if you want to debug the executable with a debugger at the source code level (cf. Chapter 7 on page 65).
- **-Wall**: Turns on lots of warning messages of the compiler. Despite its name, this flag does not enable all possible warning messages, because there is
- **-Wextra**: which turns on additional ones.
- **-Werror**: Treats warnings as errors, i.e. stops the compilation process instead of just printing a message and continuing.
- **-O0**: Disables any optimization. This option speeds up the compilations during the development/debugging stages.
- **-pedantic**: Is picky about the language standard and issues warnings about non-standard constructs. **-pedantic-errors** treats such problems as errors instead of warnings.

5.8 PGI Compilers (Lin)

Use the `module` command to load the compilers of The Portland Group into your environment. The PGI C/C++/FORTRAN 77/FORTRAN 90 compilers can be accessed by the commands **pgcc | pgCC | pgf77 | pgf90**. Please refer to the corresponding manual pages for further information. Extensible documentation is available on The Portland Group's website.[44] The following options provide a good starting point for producing well-performing machine code with these compilers:

- **-fastsse**: Turns on high optimization including vectorization.
- **-Mconcur** (compiler and linker option): Turns on auto-parallelization.
- **-Minfo**: Makes the compiler emit informative messages including those about successful and failed attempts to vectorize and/or auto-parallelize code portions.
- **-mp** (compiler and linker option): Turns on OpenMP.

Of those PGI compiler versions installed on our HPC-Cluster, the 10.x releases include support for Nvidia's CUDA architecture via the PGI Accelerator directives and CUDA FORTRAN. The following options enable this support and must be supplied during compile and link steps. (The option **-Minfo** described above is helpful for CUDA code generation, too.)

- **-ta=nvidia**: Enables PGI accelerator code generation for a GPU supporting Compute Capability 1.3 or higher.
- **-ta=nvidia,cc11**: Enables PGI accelerator code generation for a GPU supporting Compute Capability 1.1 or higher.
- **-Mcuda**: Enables CUDA FORTRAN for a GPU supporting Compute Capability 1.3 or higher.
- **-Mcuda=cc11**: Enable CUDA FORTRAN for a GPU supporting Compute Capability 1.1 or higher.

In order to read or write big-endian binary data in FORTRAN programs you can use the compiler option **-Mbyteswapio**. You can use the option **-Ktrap** when compiling the main function/program in order to enable error trapping. For information about shared memory parallelization with the PGI compilers see chapter 6.1.6 on page 58.

The PGI compiler offers several options to help you find problems with your code:

- **-g**: Puts debugging information into the object code. This option is necessary if you want to debug the executable with a debugger at the source code level (cf. Chapter 7 on page 65).
- **-O0**: Disables any optimization. This options speeds up the compilations during the development/debugging stages.
- **-w**: Disable warning messages.

5.9 Microsoft Visual Studio (Win)

Visual Studio offers a set of development tools, including an IDE (Integrated Development Environment) and support for the programming languages C++, C#, Visual Basic and Java. The current release version of Visual Studio is Visual Studio 2008. The Intel C/C++ and FORTRAN compilers are integrated into Visual Studio and can be used as well.

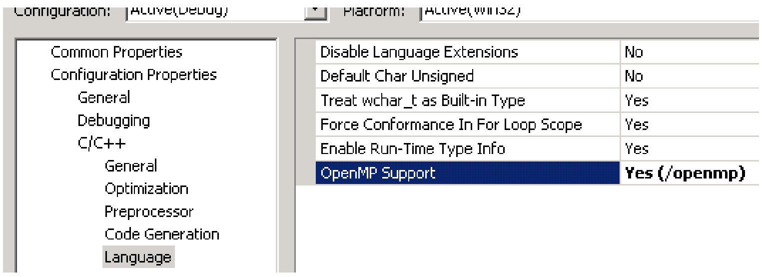
[44] http://www.pgroup.com

If you have an existing Visual Studio Project and want to use the Intel compiler, the project has to be converted to an Intel project. This can be done by right-clicking the project and selecting the lowest context menu item *Use Intel C++* ...

To change the project options, e.g. compiler or linker options, open the Project Settings window by right-clicking on the project and selecting the context menu item *Properties*. To add additional compiler options, select the compile menu (Fortran or C/C++) and add the options under *Command Line*. Here are all used compiler options listed. The most common options can also be selected in the rest of the menu.

OpenMP support can be enabled in the Project Settings window in the language options tab.

Please note that when using Visual Studio 2008 with your existing projects, these will automatically be converted and cannot be used with Visual Studio 2005 anymore. We strongly recommend making a backup of your old projects before you use Visual Studio 2008 for the first time.



5.10 Time measurements

For real-time measurements, a high-resolution timer is available. However, the measurements can supply reliable, reproducible results only on an (almost) empty machine. Make sure you have enough free processors available on the node. The number of processes which are ready to run[45] plus the number of processors needed for the measurement has to be less or equal to the number of processors. On ccNUMA CPU's like Nehalem or Opteron, be aware about processor placement and binding, see 3.1.1 on page 17.

User CPU time measurements have a lower precision and are more time-consuming. In case of parallel programs, real-time measurements should be preferred anyway!

The *r_lib* library offers two timing functions, **r_rtime** and **r_ctime**. They return the real time and the user CPU time as double precision floating point numbers. For information on how to use r_lib see 9.8 on page 89.

Depending on the operating system, programming language, compiler or parallelization paradigm, different functions are offered to measure the time. To get a listing of the file you can use

```
$ cat $PSRC/include/realtime.h
```

If you are using OpenMP the *omp_get_wtime()* function is used in background, and for MPI the *MPI_Wtime()* function. Otherwise some operating system dependent functions are selected by the corresponding C preprocessor definitions. The time is measured in seconds as double precision floating point number.

Alternatively, you can use all the different time measurement functions directly.

Linux example in C:

```
#include <sys/time.h>
struct timeval tv;
double second;
gettimeofday(&tv, (struct timezone*)0);
second = ((double)tv.tv_sec + (double)tv.tv_usec / 1000000.0);
```

In Fortran you also can use the **gettimeofday** Linux function, but it must be wrapped. Example is given in listings 12 on page 51 and 13 on page 51. After the C wrapper and the Fortran code are compiled, link and let the example binary run:

```
$ $FC rwthtime.o use_gettimeofday.o
$ ./a.out
```

[45] You can use the **uptime** command on Linux to check the load

Listing 12: $CC -c $PSRC/psr/rwthtime.c

```c
#include <sys/time.h>
#include <stdio.h>
/* This timer returns current clock time in seconds. */
double rwthtime_() {
   struct timeval tv;
   int ierr;
   ierr = gettimeofday(&tv, NULL) ;
   if (ierr != 0 ) printf("gettimeofday ERR:, ierr=%d\n",ierr);
   return ((double)tv.tv_sec + (double)tv.tv_usec / 1000000.0);
}
```

Listing 13: $FC -c $PSRC/psr/use_gettimeofday.f90

```fortran
PROGRAM t1
IMPLICIT NONE
REAL*8 rwthtime
WRITE (*,*) "Wrapped gettimeofday: ", rwthtime()
END PROGRAM t1
```

The Oracle Studio compiler has a built-in time measurement function *gethrtime*. Linux FORTRAN example with Oracle Studio compiler:

```fortran
INTEGER*8 gethrtime
REAL*8 second
second = 1.d-9 * gethrtime()
```

In FORTRAN, there is an intrinsic time measurement function called **SYSTEM_CLOCK**. The time value returned by this function can overflow, so take care about it.

The following code can be used on the Windows platform to get a high-precision low-overhead real timer:

```c
#include <Windows.h>
#define Li2Double(x) ((double)((x).HighPart)*4.294967296E9 + \
                              (double)((x).LowPart))
double SECOND (void)
{
   LARGE_INTEGER time, freq;
   QueryPerformanceCounter(&time);
   QueryPerformanceFrequency(&freq);
   return Li2Double(time) / Li2Double(freq);
}
```

Please be aware that by including Windows.h some unexpected side effects might occur, such as the definition of the macros min() and max(), which can conflict with some function of the C++ STL, for example.

5.11 Hardware Performance Counters

Hardware Performance Counters are used to measure how certain parts, like floating point units or caches, of a CPU or memory system are used. They are very helpful in finding performance

bottlenecks in programs.
The Opteron and Xeon processor core offers 4 programmable 48-bit performance counters.

5.11.1 Linux

At the moment we offer the following interfaces for accessing the counters:

- Intel VTune (see chapter 8.2.1 on page 74)
- Intel PTU (see chapter 8.2.2 on page 74)
- Oracle (Sun) Collector (see chapter 8.1 on page 71)
- PAPI Library (see chapter 8.7 on page 77)

Note: The default Linux kernel offers no support for hardware counters; there are, however, kernel patches and modules available that allow their use. We endeavor to provide the **perfctr** kernel patch on all Linux systems. At present, the kernel module for use with Intel VTune and PTU is available on a few specific machines.

5.11.2 Windows

At the moment we offer only Intel VTune and Intel PTU to access hardware counters on Windows; please refer to chapter 8.2.1 on page 74 and 8.2.2 on page 74 for more information.

6 Parallelization

Parallelization for computers with shared memory (SM) means the automatic distribution of loop iterations over several processors (automatic parallelization), the explicit distribution of work over the processors by compiler directives (OpenMP) or function calls to threading libraries, or a combination of those.

Parallelization for computers with distributed memory (DM) is done via the explicit distribution of work and data over the processors and their coordination with the exchange of messages (Message Passing with MPI).

MPI programs run on shared memory computers as well, whereas OpenMP programs usually do not run on computers with distributed memory. As a consequence, MPI programs can use all available processors of the HPC-Cluster, whereas OpenMP programs can use up to 24 processors of a node with an Intel Dunnington CPU. The soon-coming Intel Nehalem-EX based computers will offer up to 64 processors in a single machine.

For large applications the hybrid parallelization approach, a combination of coarse-grained parallelism with MPI and underlying fine-grained parallelism with OpenMP, might be attractive in order to efficiently use as many processors as possible.

Please note that long-running computing jobs should not be started interactively. Please use the batch system (see chapter 4.5 on page 27), which determines the distribution of the tasks to the machines to a large extent.

We offer examples using the different parallelization paradigms. Please refer to chapter 1.3 on page 9 for information how to use them.

6.1 Shared Memory Programming

OpenMP[46] is the de facto standard for shared memory parallel programming in the HPC realm. The OpenMP API is defined for FORTRAN, C, and C++ and consists of compiler directives (resp. pragmas), runtime routines and environment variables.

In the *parallel regions* of a program several *threads* are started. They execute the contained program segment redundantly until they hit a *worksharing construct*. Within this construct, the contained work (usually **do**- or **for**-loops, or **task** constructs since OMPv3.0) is distributed among the threads. Under normal conditions all threads have access to all data (shared data). But pay attention: If data, which is accessed by several threads, is modified, then the access to this data must be protected with *critical regions* or *OpenMP locks*.

Besides, *private* data areas can be used where the individual threads hold their local data. Such private data (in OpenMP terminology) is only visible to the thread owning it. Other threads will not be able to read or write private data.

Hint: In a loop that is to be parallelized the results must not depend on the order of the loop iterations! Try to run the loop backwards in serial mode. The results should be the same. This is a necessary, though not sufficient condition!

Note: In many cases, the stack area for the worker threads must be increased by changing a compiler-specific environment variable (e.g. Oracle Studio: STACKSIZE, Intel: KMP_STACKSIZE), and the stack area for the master thread must be increased with the command **ulimit -s xxx** (zsh shell, specification in kilobytes) or **limit s xxx** (C-shell, in kilobytes).

The number of threads to be started for each parallel region may be specified by the environment variable OMP_NUM_THREADS which is set to 1 per default on our HPC-Cluster. The OpenMP standard does not specify the number of concurrent threads to be started if OMP_NUM_THREADS is not set. In this case, the Oracle and PGI compilers start only a single thread, whereas the Intel and GNU compilers start as many threads as there are processors available. Please always set the OMP_NUM_THREADS environment variable to a reasonable value. We especially warn against setting it to a value greater than the number of processors

[46] http://www.openmp.org, http://www.compunity.org

available on the machine on which the program is to be run. On a loaded system fewer threads may be employed than specified by this environment variable because the dynamic mode may be used by default. Use the environment variable OMP_DYNAMIC to change this behavior.

If you want to use nested OpenMP, the environment variable OMP_NESTED=TRUE has to be set. Beginning with the OpenMP v3.0 API, the new runtime functions OMP_THREAD_LIMIT and OMP_MAX_ACTIVE_LEVELS are available that control nested behavior and obsolete all the old compiler-specific extensions. *Note*: Only few compilers support nested OpenMP at all.

6.1.1 Automatic Shared Memory Parallelization of Loops (Autoparallelization)

All the compilers which are installed on our HPC-Cluster can parallelize programs (more precisely: loops) automatically, at least in newer versions. This means that upon request they try to transform portions of serial FORTRAN/C/C++ code into a multithreaded program. Success or failure of autoparallelization depends on the compiler's ability to determine if it is safe to parallelize a (nested) loop. This often depends on the area of the application (e.g. finite differences versus finite elements), programming language (pointers and function calls may make the analysis difficult) and coding style.

The flags to turn this feature on differ among the various compilers. Please refer to the subsequent sections for compiler-specific information. The environment variable FLAGS_AUTOPAR offers a portable way to enable autoparallelization at compile/link time. For the Intel, Oracle, and PGI compilers, the number of parallel threads to start at runtime may be set via OMP_NUM_THREADS, just like for an OpenMP program. Only with the GNU compiler the number of threads is fixed at compile/link time.

Usually some manual code changes are necessary to help the compiler to parallelize your serial loops. These changes should be guided by compiler feedback; increasing the compiler's verbosity level therefore is recommended when using autoparallelization. The compiler options to do this as well as the feedback messages themselves are compiler-specific, so again, please consult the subsequent sections.

While autoparallelization tries to exploit multiple processors within a machine, automatic vectorization (cf. section 5.5 on page 39) makes use of instruction-level parallelism within a processor. Both features can be combined if the target machine consists of multiple processors equipped with vector units as it is the case on our HPC-Cluster. This combination is especially useful if your code spends a significant amount of time in nested loops and the innermost loop can successfully be vectorized by the compiler while the outermost loop can be autoparallelized. It is common to autoparallelization and autovectorization that both work on serial, i.e. not explicitly parallelized, code which usually must be re-structured to take advantage of these compiler features.

Table 6.14 on page 54 summarizes the OpenMP compiler options. For the currently loaded compiler, the environment variables FLAGS_OPENMP and FLAGS_AUTOPAR are set to the corresponding flags for OpenMP parallelization and autoparallelization, respectively, as is explained in section 5.2 on page 36.

Compiler	FLAGS_OPENMP	FLAGS_AUTOPAR
Oracle	-xopenmp	-xautopar -xreduction
Intel	-openmp	-parallel
GNU	-fopenmp (4.2 and above)	(empty)[48]
PGI	-mp -Minfo=mp	-Mconcur -Minline

Table 6.14: Overview of OpenMP and autoparallelization compiler options

[48] Although the GNU compiler has an autoparallelization option, we intentionally leave the FLAGS_AUTOPAR environment variable empty, see 6.1.5.2 on page 58.

6.1.2 Memory access pattern and NUMA

Today's modern computer systems have a NUMA architecture (see chapter 2.1.1 on page 12). The memory access pattern is crucial if a shared memory parallel application should not only run multithreaded, but also perform well on NUMA computers. The data accessed by a thread should be located locally in order to avoid performance penalties of remote memory access. A typical example for a bad bad memory access pattern is to initialize all data from *one* thread (i.e. in a serial program part) before using the data with *many* threads. Due to the standard *first-touch* memory allocation policy in current operating systems, all data initialized from one thread is placed in the local memory of the current processor node. All threads running on a different processor node have to access the data from that memory location over the slower link. Furthermore, this link may be overloaded with multiple simultaneous memory operations from multiple threads. You should initialize the in-memory data in the same pattern as it will be used during computation.

6.1.3 Intel Compilers (Lin / Win)

The Intel FORTRAN/C/C++ compilers support OpenMP via the compiler/linker option **-openmp** (**/Qopenmp** on Windows). This includes nested OpenMP and tasking, too. If OMP_NUM_THREADS is not set, an OpenMP program built with the Intel compilers starts as many threads as there are processors available. The worker threads' stack size may be set using the environment variable KMP_STACKSIZE, e.g.

```
$ KMP_STACKSIZE=megabytesM
```

Dynamic adjustment of the number of threads and support for nested parallelism is turned off by default when running an executable built with the Intel compilers. Please use the environment variables OMP_DYNAMIC and OMP_NESTED, respectively, to enable those features.

6.1.3.1 Thread binding Intel compilers provide an easy way for thread binding: Just set the environment variable KMP_AFFINITY to **compact** or **scatter**, e.g.

```
$ export KMP_AFFINITY=scatter
```

Setting it to **compact** binds the threads as closely as possible, e.g. two threads on different cores of one processor chip. Setting it to **scatter** binds the threads as far away as possible, e.g. two threads, each on one core on different processor sockets. Explicitly assigning OpenMP threads to a list of OS proc IDs is also possible with the **explicit** keyword. For details, please refer to the compiler documentation on the Intel website. The default behavior is to not bind the threads to any particular thread contexts; however, if the operating system supports affinity, the compiler still uses the OpenMP thread affinity interface to determine machine topology. To get a machine topology map, specify

```
$ export KMP_AFFINITY=verbose,none
```

6.1.3.2 Autoparallelization The autoparallelization feature of the Intel compilers can be turned on for an input file with the compiler option **-parallel** (**/Qparallel** on Windows), which must also be supplied as a linker option when an autoparallelized executable is to be built. The number of threads to be used at runtime may be specified in the environment variable OMP_NUM_THREADS, just like for OpenMP. We recommend turning on serial optimization via -O2 or -O3 when using **-parallel** to enable automatic inlining of function/subroutine calls within loops which may help in automatic parallelization.

You may use the option **-par-report** to make the compiler emit messages about loops which have been parallelized. If you want to exploit the autoparallelization feature of the Intel compilers it is also very helpful to know which portions of your code the compiler tried to parallelize, but failed. Via **-par-report3** you can get a very detailed report about the activities

of the automatic parallelizer during compilation. Please refer to the Intel compiler manuals about how to interpret the messages in such a report and how to subsequently re-structure your code to take advantage of automatic parallelization.

6.1.4 Oracle compilers (Lin)

The Oracle FORTRAN/C/C++ compilers support OpenMP via the compiler/linker option **-xopenmp**. This option may be used together with automatic parallelization (enabled by **-xautopar**), but loops within OpenMP parallel regions are no longer subject to autoparallelization.

The **-xopenmp** option can be used as an abbreviation for a multitude of options; the FORTRAN 95 compiler for example expands it to **-mp=openmp -explicitpar -stackvar -D_OPENMP -O3**. Please note that all local data of subroutines called from within parallel regions is put onto the stack. A subroutine's stack frame is destroyed upon exit from the routine. Therefore local data is not preserved from one call to the next. As a consequence, FORTRAN programs must be compiled with the **-stackvar** option.

The behavior of unused worker threads between parallel regions can be controlled with the environment variable SUNW_MP_THR_IDLE. The possible values are **spin |sleep |*n*s |*n*ms**. The worker threads wait either actively (busy waiting) and thereby consume CPU time, or passively (idle waiting) and must then be woken up by the system or, in a combination of these methods, they actively wait (spin) and are put to sleep ***n*** seconds or milliseconds later. With fine-grained parallelization, active waiting, and with coarse-grained parallelization, passive waiting is recommended. Idle waiting might be advantageous on an (over)loaded system.

Note: The Oracle compilers' default behavior is to put idle threads to sleep after a certain time out. Those users that prefer the old behavior (before Studio 10), where idle threads spin, can use SUNW_MP_THR_IDLE=spin to change the behavior. Please be aware that having threads spin will unnecessarily waste CPU cycles.

Note: The environment variable SUNW_MP_GUIDED_WEIGHT can be used to set the weighting value used by **libmtsk** for-loops with the *guided* schedule. The libmtsk library uses the following formula to compute the chunk sizes for *guided* loops:
`chunk_size=num_unassigned_iterations/(weight*num_threads)`
where *num_unassigned_iterations* is the number of iterations in the loop that have not yet been assigned to any thread, *weight* is a floating-point constant (default 2.0) and *num_threads* is the number of threads used to execute the loop. The value specified for SUNW_MP_GUIDED_WEIGHT must be a positive, non-zero floating-point constant.

We recommend to set SUNW_MP_WARN=TRUE while developing, in order to enable additional warning messages of the OpenMP runtime system. Do not, however, use this during production because it has performance and scalability impacts.

We also recommend the use of the option **-vpara** (FORTRAN) or **-xvpara** (C), which might allow the compiler to catch errors regarding incorrect explicit parallelization at compile time. Furthermore the option **-xcommonchk** (FORTRAN) can be used to check the consistency of *thread-private* declarations.

6.1.4.1 Thread binding

The SUNW_MP_PROCBIND environment variable can be used to bind threads in an OpenMP program to specific virtual processors (denoted with logical IDs). The value specified for SUNW_MP_PROCBIND can be one of the following:

- The string **true** or **false**
- A list of one or more non-negative integers separated by one or more spaces
- Two non-negative integers, **n1** and **n2**, separated by a minus ("-"); n1 must be less than or equal to n2 (means "all IDs from n1 to n2")

Logical IDs are consecutive integers that start with **0**. If the number of virtual processors available in the system is **n**, then their logical IDs are **0, 1, ..., n-1**.

Note: The thread binding with SUNW_MP_PROCBIND currently does not care about binding in operating system e.g. by **taskset**. This may lead to unexpected behavior or errors if using both ways to bind the threads simultaneously.

6.1.4.2 Automatic Scoping The Oracle compiler offers a highly interesting feature, which is not part of the current OpenMP specification, called *Automatic Scoping*. If the programmer adds one of the clauses *default(__auto)* or *__auto(list-of-variables)* to the OpenMP *parallel* directive, the compiler will perform the data dependency analysis and determine what the scope of all the variables should be, based on a set of scoping rules. The programmer no longer has to declare the scope of all the variables (*private*, *firstprivate*, *lastprivate*, *reduction* or *shared*) explicitly, which in many cases is a tedious and error-prone work. In case the compiler is not able to determine the scope of a variable, the corresponding parallel region will be serialized. However, the compiler will report the result of the autoscoping process so that the programmer can easily check which variables could not be automatically scoped and add suitable explicit scoping clauses for just these variables to the OpenMP parallel directive.

Add the compiler option **-vpara** to get warning messages and a list of variables for which autoscoping failed. Add the compiler option **-g** to get more details about the effect of autoscoping with the **er_src** command.

```
$ $PSRC/pex/610|| f90 -g -O3 -xopenmp -vpara -c $PSRC/psr/jacobi_autoscope.f95
$ $PSRC/pex/610|| er_src jacobi_autoscope.o
```

Find more information about autoscoping in http://docs.sun.com/source/817-6703/5_autoscope.html

6.1.4.3 Autoparallelization The option to turn on autoparallelization with the Oracle compilers is **-xautopar** which includes **-depend -O3** and in case of FORTRAN also **-stackvar**. In case you want to combine autoparallelization and OpenMP[49], we strongly suggest using the **-xautopar -xopenmp** combination. With the option **-xreduction**, automatic parallelization of reductions is also permitted, e.g. accumulations, dot products etc., whereby the modification of the sequence of the arithmetic operation can cause different rounding error accumulations.

Compiling with the option **-xloopinfo** makes the compiler emit information about the parallelization. If the number of loop iterations is unknown during compile time, code is produced which decides at runtime whether a parallel execution of the loop is more efficient or not (alternate coding). With automatic parallelization it is furthermore possible to specify the number of used threads by the environment variable OMP_NUM_THREADS.

6.1.4.4 Nested Parallelization The Oracle compilers' OpenMP support includes nested parallelism. You have to set the environment variable OMP_NESTED=TRUE or call the runtime routine **omp_set_nested()** to enable nested parallelism.

Oracle Studio compilers support the OpenMP v3.0 as of version 12, so it is recommended to use the new functions OMP_THREAD_LIMIT and OMP_MAX_ACTIVE_LEVELS to control the nesting behavior (see the OpenMP API v3.0 specification).[50]

[49] The Oracle(Sun)-specific MP pragmas have been deprecated and are no longer supported. Thus the **-xparallel** option is obsolete now. Do not use this option.

[50] However, the older Oracle(Sun)-specific variables SUNW_MP_MAX_POOL_THREADS and SUNW_MP_MAX_NESTED_LEVELS are still supported.

- SUNW_MP_MAX_POOL_THREADS specifies the size (maximum number of threads) of the thread pool. The thread pool contains only non-user threads – threads that the libmtsk library creates. It does not include user threads such as the main thread. Setting SUNW_MP_MAX_POOL_THREADS to 0 forces the thread pool to be empty, and all parallel regions will be executed by one thread. The value specified should be a non-negative integer. The default value is 1023. This environment variable can prevent a single process from creating too many threads. That might happen e.g. for recursively nested parallel

6.1.5 GNU Compilers (Lin)

As of version 4.2, the GNU compiler collection supports OpenMP with the option **-fopenmp**. The OpenMP v3.0 support is as of version 4.4 included.

The default thread stack size can be set with the variable GOMP_STACKSIZE (in kilobytes). For more information on GNU OpenMP project refer to web pages:
http://gcc.gnu.org/projects/gomp/
http://gcc.gnu.org/onlinedocs/libgomp/

6.1.5.1 Thread binding CPU binding of the threads can be done with the GOMP_CPU_AFFINITY environment variable. The variable should contain a space- or comma-separated list of CPUs. This list may contain different kind of entries: either single CPU numbers in any order, a range of CPUs (M-N), or a range with some stride (M-N:S). CPU numbers are zero-based. For example, GOMP_CPU_AFFINITY="0 3 1-2 4-15:2" will bind the initial thread to CPU 0, the second to CPU 3, the third to CPU 1, the fourth to CPU 2, the fifth to CPU 4, the sixth through tenth to CPUs 6, 8, 10, 12, and 14 respectively and then start assigning back to the beginning of the list. GOMP_CPU_AFFINITY=0 binds all threads to CPU 0. A defined CPU affinity on startup cannot be changed or disabled during the runtime of the application.

6.1.5.2 Autoparallelization Since version 4.3 the GNU compilers are able to parallelize loops automatically with the option **-ftree-parallelize-loops=<*threads*>**. However, the number of threads to use has to be specified at compile time and cannot be changed at runtime.

6.1.5.3 Nested Parallelization OpenMP nesting is supported using the standard OpenMP environment variables. *Note*: The support for OpenMP v3.0 nesting features is available as of version 4.4 of GCC compilers.

6.1.6 PGI Compilers (Lin)

To build an OpenMP program with the PGI compilers, the option **-mp** must be supplied during compile and link steps. Explicit parallelization via OpenMP compiler directives may be combined with automatic parallelization (cf. 6.1.6.2 on page 59), although loops within parallel OpenMP regions will not be parallelized automatically. The worker thread's stack size can be increased via the environment variable **MPSTKZ=*megabytes*M**.

Threads at a barrier in a parallel region check a semaphore to determine if they can proceed. If the semaphore is not free after a certain number of tries, the thread gives up the processor for a while before checking again. The MP_SPIN variable defines the number of times a thread checks a semaphore before idling. Setting MP_SPIN to **-1** tells the thread never to idle. This can improve performance but can waste CPU cycles that could be used by a different process if the thread spends a significant amount of time before a barrier.

Note: Nested parallelization is NOT supported.[51]

Note: The environment variables OMP_DYNAMIC does not have any effect.[52]

regions.

- SUNW_MP_MAX_NESTED_LEVELS specifies the maximum depth of active parallel regions. Any parallel region that has an active nested depth greater than SUNW_MP_MAX_NESTED_LEVELS will be executed by a single thread. The value should be a positive integer. The default is 4. The outermost parallel region has a depth level of 1.

[51] Refer to p. 296 (p. 322 in the PDF file) in http://www.pgroup.com/doc/pgifortref.pdf
All other shared-memory parallelization directives have to occur within the scope of a parallel region. Nested PARALLEL... END PARALLEL directive pairs are not supported and are ignored.

[52] Refer to p. 308 (p. 334 in the PDF file) ibidem.

Note: OpenMP v3.0 standard is supported, including all the nesting-related routines. However, due to lack of nesting support, these routines are dummies only.

For more information refer to http://www.pgroup.com/resources/openmp.htm or http://www.pgroup.com/resources/docs.htm.

6.1.6.1 Thread binding The PGI compiler offers some support for NUMA architectures with the option **-mp=numa**. Using NUMA can improve performance of some parallel applications by reducing memory latency. Linking **-mp=numa** also allows to use the environment variables MP_BIND, MP_BLIST and MP_SPIN. When MP_BIND is set to *yes*, parallel processes or threads are bound to a physical processor. This ensures that the operating system will not move your process to a different CPU while it is running. Using MP_BLIST, you can specify exactly which processors to attach your process to. For example, if you have a quad socket dual core system (8 CPUs), you can set the blist so that the processes are interleaved across the 4 sockets (MP_BLIST=2,4,6,0,1,3,5,7) or bound to a particular (MP_BLIST=6,7).

6.1.6.2 Autoparallelization Just like the Intel and Oracle compilers, the PGI compilers are able to parallelize certain loops automatically. This feature can be turned on with the option **-Mconcur[=*option*[,*option*,...]]** which must be supplied at compile and link time.

Some options of the **-Mconcur** option are:

- **bind** Binds threads to cores or processors.
- **levels:*n*** Parallelizes loops nested at most **n** levels deep (the default is **3**).
- **numa|nonuma** Uses (doesn't use) thread/processor affinity for NUMA architectures. **-Mconcur=numa** will link in a **numa** library and objects to prevent the operating system from migrating threads from one processor to another.

Compiler feedback about autoparallelization is enabled with **-Minfo**. The number of threads started at runtime may be specified via OMP_NUM_THREADS or NCPUS. When the option **-Minline** is supplied, the compiler tries to inline functions, so even loops with function calls may be successfully parallelized automatically.

6.2 Message Passing with MPI

MPI (Message-Passing Interface) is the de-facto standard for parallelization on distributed memory parallel systems. Multiple processes explicitly exchange data and coordinate their work flow. MPI specifies the interface but not the implementation. Therefore, there are plenty of implementations for PCs as well as for supercomputers. There are free implementations available as well as commercial ones, which are particularly tuned for the target platform. MPI has a huge number of calls, although it is possible to write meaningful MPI applications just employing some 10 of these calls.

Like the compiler environment flags, which were set by the compiler modules, we also offer MPI environment variables in order to make it easier to write platform-independent makefiles. However, these variables are only available on our Linux systems. Since the compiler wrappers and the MPI libraries relate to a specific compiler, a compiler module has to be loaded *before* the MPI module.

Some MPI libraries do not offer a C++ or a Fortran 90 interface for all compilers, e.g. the Intel MPI does not offer such interfaces for the Oracle compiler. If this is the case there will be an info printed while loading the MPI module.

- MPIEXEC – The MPI command used to start MPI applications, e.g. mprun or mpiexec.
- MPIFC, MPICC, MPICXX – Compiler driver for the last-loaded compiler module, which automatically sets the include path and also links the MPI library automatically.

- FLAGS_MPI_BATCH – Options necessary for executing in batch mode .

This example shows how to use the variables.
$ $PSRC/pex/620|| $MPICXX -I$PSRC/cpmp $PSRC/cpmp/pi.cpp -o *a.out*
$ $PSRC/pex/620|| $MPIEXEC -np 2 *a.out*

6.2.1 Interactive mpiexec wrapper (Lin)

On Linux we offer dedicated machines for interactive MPI tests. These machines will be used automatically by our interactive **mpiexec** and **mpirun** wrapper. The goal is to avoid overloading the frontend machines with MPI tests and to enable larger MPI tests with more processes.

The interactive wrapper works transparently so you can start your MPI programs with the usual MPI options. In order to make sure that MPI programs do not hinder each other the wrapper will check the load on the available machines and choose the least loaded ones. The chosen machines will get one MPI process per available processor. However, this default setting may not work for jobs that need more memory per process than there is available per core. Such jobs have to be spread to more machines. Therefore we added the **-m** *<processes per node>* option, which determines how many processes should be started per node. You can get a list of the mpiexec wrapper options with
$ mpiexec --help
which will print the list of mpiexec wrapper options, some of which are shown in table 6.15 on page 60, followed by help of native mpiexec of loaded MPI module.

--help | -h	prints this help and the help information of normal mpiexec
--show | -v	prints out which machines are used
-d	prints debugging information about the wrapper
--mpidebug	prints debugging information of the MPI lib, only OpenMPI, needs TotalView
-n, -np <np>	starts <np> processes
-m <nm>	starts exactly <nm> processes on every host (except the last one)
-s, --spawn <ns>	number of processes that can be spawned with MPI_spawn; (np+ns) processes can be started in total
--listcluster	prints out all available clusters
--cluster <clname>	uses only cluster <clname>
--onehost	starts all processes on one host
--listonly	just writes the machine file, without starting the program
MPIHOSTLIST	specifies which file contains the list of hosts to use; if not specified, the default list is taken
MPIMACHINELIST	if --listonly is used, this variable specifies the name of the created host file, default is $HOME/host.list
--skip (<cmd>)	*(advanced option)* skips the wrapper and executes the <cmd> with given arguments. Default <cmd> with sunmpi and openmpi is **mpiexec** and with intelmpi is **mpirun**.

Table 6.15: The options of the interactive mpiexec wrapper

Passing environment variables from the master, where the MPI program is started, to the other hosts is handled differently by the MPI implementations.

We recommend that if your program depends on environment variables, you let the master MPI process read them and broadcast the value to all other MPI processes.

The following sections show how to use the different MPI implementations without those predefined module settings.

6.2.2 Oracle Message Passing Toolkit and OpenMPI (Lin)

OpenMPI (www.openmpi.org) is developed by several groups and vendors, with Oracle being one of them. The Oracle Message Passing Toolkit is based on OpenMPI and thus the options and commands are the same. Both products offer very low latency communication and are the fastest MPI implementation we have installed.

Currently the Oracle Message Passing Toolkit in version 8.2.1 is the default MPI in the cluster environment, so the corresponding module is loaded by default.

To set up the environment for the Oracle Message Passing Toolkit use
`$ module load sunmpi`
and for OpenMPI use
`$ module load openmpi`
This will set environment variables for further usage. The list of variables can be obtained with
`$ module help sunmpi`

The compiler drivers are **mpicc** for C, **mpif77** and **mpif90** for FORTRAN, **mpicxx** and **mpiCC** for C++. To start MPI programs, **mpiexec** is used.

We strongly recommend using the environment variables $MPIFC, $MPICC, $MPICXX and $MPIEXEC set by the module system in particular because the compiler driver variables are set according to the latest loaded compiler module.

Refer to the manual page for a detailed description of mpiexec. It includes several helpful examples.

For quick reference we include some options here, see table 6.16 on page 61.

Option	Description
-n <#>	Number of processes to start.
-H <**host1,..,hostN**>	Synonym for **-host**. Specifies a list of execution hosts.
-machinefile <**machinefile**>	Where to find the machinefile with the execution hosts.
-mca <**key**> <**value**>	Option for the Modular Component Architecture. This option e.g. specifies which network type to use.
-nooversubscribe	Does not oversubscribe any nodes.
-nw	Launches the processes and do not wait for their completion. mpiexec will complete as soon as successful launch occurs.
-tv	Launches the MPI processes under the TotalView debugger (old style MPI launch, currently doesn't work with Oracle MPI and Intel compiler; please use the new style MPI launch).
-wdir <**dir**>	Changes to the directory <dir> before the user's program executes.
-x <**env**>	Exports the specified environment variables to the remote nodes before executing the program.

Table 6.16: OpenMPI mpiexec options

6.2.3 Intel's MPI Implementation (Lin)

Intel provides a commercial MPI library based on **MPICH2** from Argonne National Labs. It may be used as an alternative to Oracle MPI.

On Linux, Intel MPI can be initialized with the command
`$ module switch sunmpi intelmpi`
This will set up several environment variables for further usage. The list of these variables can be obtained with
`$ module help intelmpi`
In particular, the compiler drivers **mpiifort**, **mpifc**, **mpiicc**, **mpicc**, **mpiicpc** and **mpicxx** as well as the MPI application startup scripts **mpiexec** and **mpirun** are included in the search path.[53] The compiler drivers **mpiifort**, **mpiicc** and **mpiicpc** use the Intel Compilers whereas **mpifc**, **mpicc** and **mpicxx** are the drivers for the GCC compilers. The necessary include directory $MPI_INCLUDE and the library directory $MPI_LIBDIR are selected automatically by these compiler drivers.

We strongly recommend using the environment variables $MPIFC, $MPICC, $MPICXX and $MPIEXEC set by the module system for building and running an MPI application. Example:

```
$ $MPIFC -c prog.f90
$ $MPIFC prog.o -o prog.exe
$ $MPIEXEC -np 4 prog.exe
```

The Intel MPI can basically be used in the same way as the Oracle MPI, except of the Oracle-specific options, of course. You can get a list of options specific to the startup script of Intel MPI by
`$ $MPIEXEC -h`

If you want to use the compiler drivers and startup scripts directly, you can do this as shown in the following examples.

Example using an MPI compiler wrapper for the Intel FORTRAN compiler:

```
$ mpiifort -c prog.f90
$ mpiifort -o prog.exe prog.o
$ mpiexec -np 4 prog.exe
```

Example using the Intel FORTRAN compiler directly:

```
$ ifort -I$MPI_INCLUDE -c prog.f90
$ ifort prog.o -o prog.exe -L$MPI_LIBDIR -lmpi
$ mpiexec -np 4 prog.exe
```

6.2.4 Microsoft MPI (Win)

Microsoft MPI is based on **mpich2**. To use Microsoft MPI, you have to prepare your build environment for compilation and linking. You have to provide *C:\Program Files\Microsoft HPC Pack 2008 SDK\Include* as an include directory during compile time. These are the directories for the headers (mpi.h for C/C++ programs and mpif.h for FORTRAN programs). Additionally there is a FORTRAN 90 module available with mpi.f90. You also have to provide *C:\Program Files\Microsoft HPC Pack 2008 SDK\Lib\[i386/AMD64]* as an additional library directory. To create 32bit programs, you have to choose the subdirectory i386, for 64bit programs you have to choose AMD64. The required library is msmpi.lib which you have to link.

To add the paths and files to Visual Studio, open your project properties (*"Project"* → *"Properties"*) and navigate to *"C/C++"* or *"Fortran"* → *"General"* for the include directory, *"Linker"* → *"General"* for the library directory and *"Linker"* → *"Input"* for the libraries.

[53] Currently, these are not directly accessible, but obscured by the wrappers we provide.

6.3 Hybrid Parallelization

The combination of MPI and OpenMP and/or autoparallelization is called *hybrid parallelization*. Each MPI process may be multi-threaded. In order to use hybrid parallelization the MPI library has to support it. There are four stages of possible support:

1. single – multi-threading is not supported.
2. funneled – only the main thread, which initializes MPI, is allowed to make MPI calls.
3. serialized – only one thread may call the MPI library at a time.
4. multiple – multiple threads may call MPI, without restrictions.

You can use the **MPI_Init_thread** function to query multi-threading support of the MPI implementation. Read more on this web page:
http://www.mpi-forum.org/docs/mpi22-report/node260.htm

In listing 14 on page 63 an example program is given which demonstrates the switching between threading support levels in case of a Fortran program. This program can be used to test if a given MPI library supports threading.

Listing 14: $MPIFC $PSRC/pis/mpi_threading_support.f90; a.out

```
PROGRAM tthr
USE MPI
IMPLICIT NONE
INTEGER :: REQUIRED, PROVIDED, IERROR
REQUIRED = MPI_THREAD_MULTIPLE
PROVIDED = -1
!  A call to MPI_INIT has the same effect as a call to
!  MPI_INIT_THREAD with a required = MPI_THREAD_SINGLE.
!CALL MPI_INIT(IERROR)
CALL MPI_INIT_THREAD(REQUIRED, PROVIDED, IERROR)
WRITE (*,*) MPI_THREAD_SINGLE, MPI_THREAD_FUNNELED, &
& MPI_THREAD_SERIALIZED, MPI_THREAD_MULTIPLE
WRITE (*,*) REQUIRED, PROVIDED, IERROR
CALL MPI_FINALIZE(IERROR)
END PROGRAM tthr
```

6.3.1 Oracle Message Passing Toolkit and OpenMPI (Lin)

Oracle MPI version 8.x is a release of OpenMPI. The OpenMPI community site announces untested support for thread-safe operations.[54] The support for threading is disabled by default in OpenMPI and was not enabled by Oracle, thus Oracle MPI version 8.x does not support thread-safe operations: only *MPI serialized* is supported.[55]

We provide some versions of OpenMPI with threading support enabled.[56] However, due to less-tested status of this feature, use it at own risk.

6.3.2 Intel-MPI (Lin)

Unfortunately, *Intel-MPI* is not thread-safe by default. To provide full MPI support inside parallel regions the program must be linked with the option *-mt_mpi*.
Note: If you specify one of the following options for the Intel FORTRAN Compiler, the thread-safe version of the library is used automatically:

[54] http://www.open-mpi.org/faq/?category=supported-systems#thread-support
[55] http://docs.sun.com/source/821-0840-10/body.html#50581507_pgfId-1006645
[56] Configured and compiled with *–enable-mpi-threads* option.

1. -openmp
2. -parallel
3. -threads
4. -reentrancy
5. -reentrancy threaded

The **funneled** level is provided by default by the thread-safe version of the Intel MPI library. To activate other levels, use the **MPI_Init_thread** function.

6.3.3 Microsoft MPI (Win)

Microsoft MPI currently supports MPI serialized.

7 Debugging

If your program is having strange problems there's no need for immediate despair - try leaning back and thinking hard first:

- Which were the latest changes that you made? (A source code revision system e.g. SVN, CVS or RCS might help.)
- Reduce the optimization level of your compilation.
- Choose a smaller data set. Try to build a specific test case for your problem.
- Look for compiler messages and warnings.
- Use tools for a static program analysis (see chapter 7.1 on page 65).
- Try a dynamic analysis with appropriate compiler options (see chapter 7.2 on page 66).
- Reduce the number of CPUs in a parallel program; try a serial program run, if possible.
- Use a debugger like TotalView (see chapter 7.3.1 on page 67). Use the smallest case which shows the error.
- In case of an OpenMP program, use a thread-checking tool like the Oracle Thread Analyzer (see chapter 7.4.1 on page 68) or the Intel ThreadChecker (see chapter 7.4.2 on page 69).
- If it is an OpenMP program, try to compile without optimization, e.g. with **-g -O0 -xopenmp=noopt** for the Oracle compilers.
- In case of an MPI program, use an MPI tool (see chapter 8.8 on page 78) or a parallel debugger like TotalView. Try another MPI implementation version and/or release.
- Try a different compiler. Maybe you have run into a compiler bug?

7.1 Static Program Analysis

First, an exact static analysis of the program is recommended for error detection. Today's compilers are quite smart and can detect many problems. Turn on a high verbosity level while compiling and watch for compiler warnings. Please refer to Chapter 5 for various compiler options regarding warning levels.

Furthermore, the tools listed in table 7.17 on page 65 can be used for static analysis.

lint	syntax check of C programs (distributed with Oracle Studio compilers)
ftnchek	syntax check of FORTRAN 77 programs (with some FORTRAN 90 features)
cppcheck	syntax check of C++ programs. http://sourceforge.net/projects/cppcheck/

Table 7.17: Static program analysis tools (Lin)

Sometimes program errors occur only with high (or low) compiler optimization. This can be a compiler error or a program error. If the program runs differently with and without compiler optimizations, the module causing the trouble can be found by systematic bisecting. With this technique, you compile half of the application with the *right* options and the other half with the *wrong* options. If the program then fails, you will know which part causes the problem. Likewise, if the program runs fine afterwards, repeat the process for the part of the program causing the failure.

7.2 Dynamic Program Analysis

Many compilers offer options to perform runtime checks of the generated program, e.g. array bound checks or checks for uninitialized variables. Please study the compiler documentation and look for compiler options which enable additional runtime checks. Please note that such checks usually cause a slowdown of your application, so do not use them for production runs.

The **Intel Fortran** compiler allows you to turn on various runtime checks with the **-check** flag. You may also enable only certain conditions to be checked (e.g. **-check bounds**), please consult the compiler manual for available options.

The **Oracle Fortran** compiler does array bound checking with the option **-C** and global program analysis with the option **-Xlist**. Compiling with **-xcheck=init_local** initializes local variables to a value that is likely to cause an arithmetic exception if it is used before it is assigned by the program. Memory allocated by the ALLOCATE statement will also be initialized in this manner. SAVE variables, module variables, and variables in COMMON blocks are not initialized. Floating point errors like division by zero, overflows and underflows are reported with the option **-ftrap=%all**.

The Oracle compilers also offer the option **-xcheck=stkovf** to detect stack overflows at runtime. In case of a stack overflow a core file will be written that can then be analyzed by a debugger. The stack trace will contain a function name indicating the problem.

The **GNU C/C++** compiler offers the option **-fmudflap** to trace memory accesses during runtime. If an illegal access is detected, the program will halt. With **-fbounds-check** the array bound checking can be activated.

To detect *common errors with dynamic memory allocation*, you can use the library **libefence** (Electric Fence). It helps to detect two common programming bugs: software that overruns the boundaries of a *malloc()* memory allocation, and software that touches a memory allocation that has been released by *free()*. If an error is detected, the program stops with a segmentation fault and the error can easily be found with a debugger. To use the library, link with **-lefence**. For more information see the manual page (**man libefence**).

Memory leaks can be detected using TotalView (see chapter A.1.8 on page 94), the sampling collector (**collect -H**, see chapter 8.1 on page 71) or the open source instrumentation framework Valgrind (please refer to http://valgrind.org).

If a program with optimization delivers other results than without, floating point optimization may be responsible. There is a possibility to test this by optimizing the program carefully. Please note that the environment variables $FLAGS_FAST and $FLAGS_FAST_NO_FPOPT containing different sets of optimization flags for the last-loaded compiler module. If you use $FLAGS_FAST_NO_FPOPT flag instead of $FLAGS_FAST, the sequence of the floating point operations is not changed by the optimization, perhaps increasing the runtime.

Besides, you have to consider that on the x86 platform floating point calculations do not necessarily conform to IEEE standard by default, so rounding effects may differ between platforms.

7.3 Debuggers

A Debugger is a tool to control and look into a running program. It allows a programmer to follow the program execution step by step and see e.g. values of variables. It is a powerful tool for finding problems and errors in a program.

For debugging, the program must be translated with the option **-g** and optimization should be turned off to facilitate the debugging process. If compiled with optimization, some variables may not be visible while debugging and the mapping between the source code and the executable program may not be accurate.

A core dump can be analyzed with a debugger, if the program was translated with **-g**. Do not forget to increase the core file size limit of your shell if you want to analyze the core that

your program may have left behind.

```
$ ulimit -c unlimited
```

But please do not forget to purge core files afterwards!

Note: You can easily find all the core files in your home dir with the following command:

```
$ find $HOME -type f -iname "core*rz.RWTH-Aachen.DE*"
```

In general we recommend using a full-screen debugger like TotalView or Oracle Studio to

- start your application and step through it,
- analyze a core dump of a prior program run,
- attach to a running program.

In some cases, e.g. in batch scripts or when debugging over a slow connection, it might be preferable to use a line mode debugger like **dbx** or **gdb**.

7.3.1 TotalView (Lin)

The state-of-the-art debugger TotalView from TotalView Technologies[57] can be used to debug serial and parallel FORTRAN, C and C++ programs. You can choose between different versions of TotalView with the module command.[58] From version 8.6 on, TotalView comes with the ReplayEngine. The ReplayEngine allows backward debugging or reverting computations in the program. This is especially helpful if the program crashed or miscomputed and you want to go back and find the cause.

In the appendix A on page 92 we include a TotalView Quick Reference Guide.

We recommend a watchful study of the User's Manual[59] and the Reference Guide[60] to find out about all the near limitless skills of TotalView debugger. The module is loaded with:

```
$ module load totalview
```

7.3.2 Oracle's Integrated Development Environment (IDE) (Lin)

Oracle Studio includes a complete Integrated Development Environment (IDE). It also contains a full screen debugger for serial and multi-threaded programs. You will find an extensive online help information on how to use the environment.

In order to start a debugging session, you can attach to a running program with

```
$ module load studio
$ sunstudio -A pid
```

or analyze a core dump with

```
$ sunstudio -C core:a.out
```

or start the program under the control of the debugger with

`$ $PSRC/pex/730|| sunstudio -D a.out` At the top of the **sunstudio** window you can find the debugger controls to start, stop, pause and step through your program. Just move your mouse over the buttons to display their function. The upper right frame shows the source of your program. You can set and clear breakpoints by clicking on the gray bar left of the corresponding source line. Output of your program is displayed and input can be typed in the lower right frame if you selected the *Process I/O* pane. You may want to switch to the *Debugger Console* in order to see the debugger messages in the same frame.

[57] Etnus was renamed to TotalView Technologies: http://www.totalviewtech.com/
[58] see chapter 4.4.2 on page 25
[59] http://www.totalviewtech.com/pdf/other/TotalView_User_Guide.pdf
[60] http://www.totalviewtech.com/pdf/other/totalview_reference_guide.pdf

In the frame on the left part of the *sunstudio* window, various information displays on the state of the debugged program can be seen: sessions, threads, call stack, local variables, watchpoints, breakpoints, properties.

7.3.3 Intel idb (Lin)

idb is a debugger for serial and parallel (multi-threaded and OpenMP) programs compiled with the Intel compilers. It is shipped together with the Intel compilers, so you do not need to load any additional module for idb if Intel compilers are loaded. Since the 11 compiler series, **idb** is based on eclipse and starts with a GUI by default. However, the command line debugger can still be started with the **idbc** command. The GUI requires Java and is known to be be unstable sometimes with the default Java version, so load a version of Java from the module system via

```
$  module load java
```

if you encounter problems.

Some documentation on the Intel debugger may be found here:
http://www.fz-juelich.de/jsc/docs/vendordocs/idbe/doc/idb_manual/

7.3.4 gdb (Lin / Win)

gdb is a powerful command line-oriented debugger. The corresponding manual pages as well as online manuals are available for further information.

7.3.5 pgdbg (Lin)

pgdbg is a debugger with a GUI for debugging serial and parallel (multithreaded, OpenMP and MPI) programs compiled with the PGI compilers.

7.3.6 Allinea ddt (Lin)

Allinea **ddt** (Distributed Debugging Tool) is a debugger with a GUI for serial and parallel programs. It can be used for multithreaded, OpenMP and MPI applications. Furthermore, since version 2.6 it can handle GPGPU programs written with NVIDIA Cuda. The module is located in the `DEVELOP` category and can be loaded with:

```
$ module load ddt
```

For full documentation please refer to: http://www.allinea.com/downloads/userguide.pdf

7.4 Runtime Analysis of OpenMP Programs

If an OpenMP program runs fine using a single thread but not multiple threads, there is probably a data sharing conflict or data race condition. This is the case if e.g. a variable which should be private is shared or a shared variable is not protected by a lock.

The presented tools will detect data race conditions during runtime and point out the portions of code which are not thread-safe.

Recommendation:
Never put an OpenMP code into production before having used a thread checking tool.

7.4.1 Oracle's Thread Analyzer (Lin)

Oracle (Sun) integrated the Thread Analyzer, a data race detection tool, into the Studio compiler suite. The program can be instrumented while compiling, so that data races can be detected at runtime. The Thread Analyzer also supports nested OpenMP programs.

Make sure you have the version 12 or higher studio module loaded to set up the environment. Add the option **-xinstrument=datarace** to your compiler command line. Since

additional functionality for thread checking is added the executable will run slower and need more memory. Run the program under the control of the **collect**[61] command

```
$ $PSRC/pex/740|| $CC $FLAGS_OPENMP -xinstrument=datarace $PSRC/C-omp-pi/pi.c
-lm $FLAGS_DEBUG
$ $PSRC/pex/740|| collect -r on a.out
```

You have to use more than one thread while executing, since only *occurring* data races are reported. The results can be viewed with **tha**, which contains a subset of the analyzer functionality, or the **analyzer**.

```
$ $PSRC/pex/740|| tha tha.1.er
```

7.4.2 Intel Thread Checker (Lin / Win)

The succeeding product of Assure is the Intel Thread Checker, which is available for Linux and Windows. It is able to verify the correctness of multithreaded programs.

The Thread Checker product has two different modes: the source instrumentation mode and the binary instrumentation mode. The source instrumentation mode can find more problems than the binary instrumentation mode, but it only supports Intel compilers and OpenMP programs. Furthermore the program has to be independent of the number of threads used, which is the case if the program does not contain any conditional branches or execution paths depending on the number of threads. Therefore it is sometimes referred to as the thread-count dependent mode.

The binary instrumentation mode may overlook a few error classes, but it is independent of the number of threads used and it supports POSIX threads and Windows threads as well. In this mode, binaries produced with other compilers than Intel may also be used (debug info provided).

Be aware that under the control of the Thread Checker, in both modes, your program will need *a lot* more memory and run *noticeably* slower, elongations of a factor 10 to 1000 are possible.

To prepare your environment on Linux, you have to use

```
$ module load intelitt
```

On Windows, the Thread Checker is only available on the cluster frontend machine.

Source Instrumentation Mode: Compile your program using the Intel compilers with **-tcheck** on Linux or **/Qtcheck** on Windows. On Linux you have to extend your command line with **$ITT_LINK** as well. On Windows you have to link with the **/fixed:no** option. Then you can run your program as usually. After the program has finished, a **threadchecker.thr** file is written that can be viewed using the **tcheck_cl** command line tool on Linux.

Linux example:

```
$ $PSRC/pex/741|| $CC -g $FLAGS_OPENMP -tcheck $PSRC/C-omp-pi/pi.c -lm $ITT_LINK
$ $PSRC/pex/741|| a.out; tcheck_cl threadchecker.thr
```

On Windows, the **threadchecker.thr** file may be opened in the Thread Checker GUI: Click *Start* →*Programs* →*Intel Software Development Tools* →*Intel Thread Checker* →*Intel Thread Checker* to start the GUI, then *Close* to disable the *Easy Start* menu, then *File* →*Open File* and then browse to the file and then *Open*.

Binary Instrumentation Mode: Make sure to compile your program with debug information generation enabled. On Linux, you have to run your program under control of Thread Checker using **tcheck_cl**. After the program finishes, a summary of the analysis is written on the console.

Linux example:

```
$ $PSRC/pex/742|| tcheck_cl a.out | tee my_tcheck_out.txt
```

[61] more details are given in the analyzer section 8.1 on page 71

On Windows, start the Thread Checker GUI: Click *Start* →*Programs* →*Intel Software Development Tools* →*Intel Thread Checker* →*Intel Thread Checker*. Then select *New Project* →*Threading Wizards* →*Intel thread Checker Wizard* →*OK* and then choose an executable file and click *Finish*. If your program runs for any length of time you wish to see the intermediate results, use the *Intel Thread Checker Realtime Status* box.

As in the Source Instrumentation Mode, it is possible to generate the **.thr** file on Linux and to open it using the Windows GUI.

Remote Real-Time Mode: It is also possible to use Thread Checker on Linux and view the results on Windows in real-time. You have to compile your program on Linux as described above and then start the server for the Thread Checker:

```
$ ittserver
```

On the windows side you have to create a new Thread Checker project: *File* →*New Project...*. Choose *Category* →*Intel Thread Checker Wizard* and specify a project name and location. In the *Intel Thread Checker Wizard* box you need to configure the *Host* (e.g., cluster.rz.rwth-aachen.de), where you started the server. Furthermore specify the path to the executable, the working directory and the command line arguments and click *Next* →*Finish*. Now your application should be started on the remote server.

Please note that the server will be started on a fixed port (50002). Only one user can start an instance of it at the same time. Please DO NOT try to connect to the server if you get an error like this:

```
[ FATAL] Unable to bind to port 50002 : Address already in use
```

In this case another user already runs a server. You can try to use another frontend or try again later. Please note that every user within the RWTH cluster could connect to the server you started. So ensure shutting down after running a sample and watch any incoming connection in the server output while it is running.

For long execution times of your application it is not recommended to make use of the remote real-time viewing feature.

More information on the Intel Thread Checker may be found in these locations:
/opt/intel/itt/<ITT_VERSION>/tcheck/doc
http://www.intel.com/support/performancetools/threadchecker/linux/index.htm

8 Performance / Runtime Analysis Tools

This chapter describes tools that are available to help you assess the performance of your code, identify potential performance problems, and locate the part of the code where most of the execution time is spent.

Runtime analysis is no trivial matter and cannot be sufficiently explained in the scope of this document. An introduction to some of the tools described in this chapter will be given at workshops in Aachen and other sites in regular intervals. If you need help using these tools, please contact the HPC group.

8.1 Oracle Sampling Collector and Performance Analyzer (Lin)

The Oracle Sampling Collector and the Performance Analyzer are a pair of tools that you can use to collect and analyze performance data for your serial or parallel application. The Collector gathers performance data by sampling at regular time intervals and by tracing function calls. The performance information is gathered in so-called experiment files, which can then be displayed with the **analyzer** GUI or the **er_print** line command after the program has finished. Since the collector is part of the Oracle compiler suite, the *studio compiler module* has to be loaded. However, programs to be analyzed do not have to be compiled with the Oracle compiler; the GNU or Intel compiler for example work as well.

8.1.1 The Oracle Sampling Collector

At first you have to compile your program with the **-g** option if you want to benefit from the full functionality of the analyzer, like the source-line attribution. When compiling C++ code with the Oracle compiler you can use the **-g0** option instead if you want to let the compiler expand inline functions for performance reasons.

Link the program as usual and then start the executable under the control of the Sampling Collector with the command

$ $PSRC/pex/810|| `collect a.out`

or with the analyzer (select *Collect Experiment* under the *File* menu)

Every 10 milliseconds profile data will be gathered and written to the experiment file **test.1.er**. The filename number will be automatically incremented on subsequent experiments. In fact the experiment file is an entire directory with a lot of information. You can manipulate these with the regular Linux commands, but it is recommended to use the *er_mv*, *er_rm*, *er_cp* utilities to move, remove or copy these directories. This ensures for example that time stamps are preserved.

By selecting the options of the collect command, many different kinds of performance data can be gathered. Just invoking **collect** will print a complete list including available hardware counters. The most important collect options are listed in table 8.18 on page 72.

Various hardware-counter event-types can be chosen for collecting. The maximum number of theoretically simultaneously usable counters on available hardware platforms ranges between 4 (AMD Barcelona) and 7 (Intel Nehalem). However, it is hardly possible to use more than 4 counters in the same measurement because some counters use the same resources and thus conflict with each other. Favorite choices are given in table 8.19 on page 72 for Barcelona CPUs, in table 8.20 on page 73 for Harpertown, Tigerton and Dunnington CPUs, and in table 8.21 on page 73 for Nehalem EP CPUs.

This example counts the floating point additions and multiplications as wells as the L1 and L2 cache misses in addition to the clock profiling on Opteron processors:

$ $PSRC/pex/811|| `collect -p on -h fpadd,on,fpmul,on,dcm,high,ecdm,on a.out`

-p on | off | hi | lo	Clock profiling ('hi' needs to be supported on the system)
-H on | off	Heap tracing
-m on | off	MPI tracing
-h *counter0*,on,...	Hardware Counters
-j on | off	Java profiling
-S on | off | *seconds*	Periodic sampling (default interval: 1 sec)
-o *experimentfile*	Output file
-d *directory*	Output directory
-g *experimentgroup*	Output file group
-L *size*	Output file size limit [MB]
-F on | off	Follows descendant processes
-C *comment*	Puts comments in the notes file for the experiment

Table 8.18: Collect options

-h cycles,on,insts,on	Cycle count, instruction count. The quotient is the CPI rate (clocks per instruction). The MHz rate of the CPU multiplied with the instruction count divided by the cycle count gives the MIPS rate. Alternatively, the MIPS rate can be obtained as the quotient of instruction count and runtime in seconds.
-h fpadd,on,fpmul,on	Floating point additions and multiplications. The sum divided by the runtime in seconds gives the FLOPS rate.
-h cycles,on,dtlbm,on,dtlbh,on	Cycle count, data translation look-aside buffer (DTLB) misses and hits. A high rate of DTLB misses indicates an unpleasant memory access pattern of the program. Large pages might help.
-h dc,on,dcm,on,ecd,on,ecdm,on	L1 and L2 cache references and misses. A high rate of cache misses indicates an unpleasant memory access pattern of the program.

Table 8.19: Hardware counter available for profiling with **collect** on AMD Barcelona CPUs

8.1.2 The Oracle Performance Analyzer

Collected experiment data can be evaluated with the **analyzer** GUI:

```
$ $PSRC/pex/810|| analyzer test.1.er
```

A program call tree with performance information can be displayed with the locally developed utility **er_view**:

```
$ $PSRC/pex/810.1|| er_view test.1.er
```

There is also a command line tool **er_print** with basically the same functionality. Invoking **er_print** without options will print a command overview. Example:

```
$ $PSRC/pex/810.2|| er_print -fsummary test.1.er | less
```

8.1.3 The Performance Tools Collector Library API

Sometimes it is convenient to group performance data in self-defined samples, and to collect performance data of a specific part of the program only. For this purpose the **libcollectorAPI** library can easily be used.

In the following example FORTRAN program, performance data of the subroutines *work1* and *work2* is collected.

-h cycles,on,insts,on	Same meaning as in table 8.19 on page 72
-h fp_comp_ops_exe,on	The count of floating point operations divided by the runtime in seconds gives the FLOPS rate.
-h cycles,on,dtlbm,on	Cycle count, data translation look-aside buffer (DTLB) misses. A high rate of DTLB misses indicates an unpleasant memory access pattern of the program. Large pages might help.
-h llc-reference,on,llc-misses,on -h l2_ld,on,l2_lines_in,on -h l1i_reads,on,l1i_misses,on	Last level cache references and misses L2 cache references and misses L1 instruction cache references and misses

Table 8.20: Hardware counter available for profiling with **collect** on Intel Harpertown, Tigerton and Dunnington CPUs

-h cycles,on,insts,on	Same meaning as in table 8.19 on page 72
-h fp_comp_ops_exe.x87,on, fp_comp_ops_exe.mmx,on, fp_comp_ops_exe.sse_fp	Floating point counters on different execution units. The sum divided by the runtime in seconds gives the FLOPS rate.
-h cycles,on,dtlb_misses.any,on	A high rate of DTLB misses indicates an unpleasant memory access pattern of the program. Large pages might help.
-h llc-reference,on,llc-misses,on -h l2_rqsts.references,on,l2_rqsts.miss,on -h l1i.hits,on,l1i.misses,on	Last level (L3) cache references and misses L2 cahce references and misses L1 instruction cache hits and misses

Table 8.21: Hardware counter available for profiling with **collect** on Intel Nehalem CPUs

Listing 15: f90 $PSRC/pis/collector.f90; a.out

```
program testCollector
    double precision :: x
    call collector_pause()
    call PreProc(x)
    call collector_resume()
    call collector_sample("Work1")
    call Work1(x)
    call collector_sample("Work2")
    call Work2(x)
    call collector_terminate_expt()
    call PostProc(x)
end program testCollector
```

The **libcollectorAPI** library (or when using FORTRAN, **libfcollector**) has to be linked. If this program is started by

`$ collect -S off a.out`

performance data is only collected between the *collector_resume* and the *collector_terminate_expt* calls. No periodic sampling is done, but single samples are recorded whenever *collector_sample* is called. When the experiment file is evaluated, the filter mechanism can be used to restrict the displayed data to the interesting program parts. The timelines display includes the names of the samples for better orientation. Please refer to the *libcollector* manual page for further information.

8.2 Intel Performance Analyze Tools (Lin / Win)

The Intel **VTune** Performance Analyzer and Intel Performance Tuning Utility (**PTU**) provide an integrated performance analysis and tuning environment. The invaluable feature of VTune and PTU is the possibility to use hardware counters. Both tools use the same backend to get information, but they represent the collected information in different ways.

For some operations you need extended permissions. Therefore, Intel VTune can only be used with the assistance of the HPC-group. Please contact us if you want to use Intel VTune and/or PTU. Due to the need of additional kernel modules for the hardware counters, which could cause some instability of the operating system, Intel VTune and PTU are installed on few machines only. The tools are designed to analyze binaries built with the Intel compilers, but other compilers can be used as well. The program has to be compiled with debug information (**-g**) to see source code.

On **Windows** you need to be added to the **g_tuning** group to get access to the tuning machines. Log in to **cluster-win-tuning.rz.RWTH-Aachen.DE** and start VTune or PTU from the *Start* menu or *Desktop*.

On **Linux** you need to be added to the **vtune** group to get access to the machines where VTune and PTU is installed. Log in to **cluster-linux-tuning.rz.RWTH-Aachen.DE** with ssh. The cluster-linux-tuning computer uses the **Kerberos** service[62] for authentication. You therefore need a valid **ticket** to gain access to network-mounted directories. If you are logged in using NX software (see chapter 4.1.2 on page 20), a ticket will typically be initialized automatically. If not, or if you are logged in without using NX, you need to obtain a valid ticket by the command **kinit** before logging in to cluster-linux-tuning. This utility will ask for your cluster password.

Note: The Kerberos ticket expires after 24 hours, so you may need to use **kinit** every day.
Note: With the **klist** utility you can check your Kerberos ticket.
Note: It is not possible to log in to **cluster-linux-tuning.rz.RWTH-Aachen.DE** directly with a NX-Client, but only over one of the cluster-x nodes.

After login, load the VTune module and start VTune:

```
$ module load intelvtune
$ vtlec
```

or load the PTU module and start the Eclipse-based GUI:

```
$ module load intelptu
$ ptugui
```

8.2.1 Intel VTune

To analyze your code, click *File* →*New* →*Project*. Then you can choose *Call Graph Wizard* to get a call graph of your program, or *Sampling Wizard* to analyze your code using hardware counters. Now mark Linux executable and click *Next*. After that, specify the application to launch, the command line arguments and a working directory. At last choose an eclipse project for your sampling run and press *Finish*.

If you have chosen *Call Graph Wizard*, a call graph of your program is shown. When *Sampling Wizard* was chosen you get sampling results for the counters *CPU_CLK_UNHALTED.CORE* and *INST_RETIRED.ANY*. To use other hardware counters click *Tuning* →*Modify <Sampling Activity>* and the left configure button. This opens a window where you can choose other counters.

8.2.2 Intel PTU

The Intel Performance Tuning Utility offers:

[62] http://tools.ietf.org/html/rfc4120

- Statistical call graph: Profiles with low overhead to detect where time is spent in your application
- Predefined profile configurations: Prepared profiles e.g. "loop analysis on Intel Nehalem CPUs"
- Data Access Profiling: Identifies memory hotspots and relates them to code hotspots
- Heap Profiler: Identifies dynamic memory usage by application. Can help identify memory leaks
- ... and many more features.

More information about intel PTU is available here: http://software.intel.com/en-us/articles/intel-performance-tuning-utility/

After you started the PTU GUI by **ptugui**, choose a place where the workspace will be stored. Then click *File* →*New* →*Project* and choose *New Intel PRU Project*, click *Next*, type a project name and click *Next*. Choose an executable file (*Application*), type application command line parameters (if there are any) and working directory. A new project will be created by clicking on *Finish*.

To perform some profiling, right-click on the new project in the left *Tuning Navigator* pane, go to *Profile as* and choose a predefined profile. The application will be launched (maybe more than one time) and after some analysis you should see a new pane with profiling results on the right.

By right-clicking on the project in the left *Tuning Navigator* pane and choosing *Profile*, a new profile can be defined.

Alongside with the GUI, the Intel Performance Tuning Utility has a diverse set of command line utilities, including displays, e.g. **vtsarun** and **vtsaview**.

8.3 Intel Thread Profiler (Lin / Win)

The Intel Thread Profiler is the second part of the Intel Threading tools, along with the Intel Thread Checker. It helps developers to analyze the thread level performance of applications that use Windows, POSIX or OpenMP threads.

The usage of the Thread Profiler is similar to the Thread Checker (see section 7.4.2 on page 69). It also has two different modes: the *source instrumentation* mode and the *binary instrumentation* mode.

To load the Intel Threading Tools module on **Linux**, you have to use

```
$ module load intelitt
```

On **Windows**, the Thread Profiler is only available on the cluster frontend machines.

Source instrumentation mode: Compile your program using the Intel compilers with **-tprofile** on Linux or **/Qtprofile** on Windows. On Linux you can use **-openmp-profile** to enable analysis of OpenMP applications.

On Linux you have to extend your command line with the **$ITT_LINK** option as well. On Windows you have to link with the **/fixed:no** option. Then run your program as usually. After the program has finished, a short overview of the run is shown and the files **tp.tp** and **tp.tpd** are written. If you enabled the analysis of OpenMP applications by **-openmp-profile**, a plain text file **guide.gvs** will be written.

Linux Example:

```
$ $PSRC/pex/831|| $CC $FLAGS_DEBUG $FLAGS_OPENMP -tprofile $PSRC/C-omr-pi/pi.c
-lm $ITT_LINK
$ $PSRC/pex/831|| a.out
```

There is only a Windows GUI to analyze these files. Start the Intel Thread Profiler from *Start* →*Programs* →*Intel Software Development Tools* →*Intel Thread Profiler* →*Intel Thread Profiler*.

In the *Easy Start* dialog box, click *Close*, as you want to view a data file, not start a Thread Profiler activity. Select *File* →*Open file* and browse to the directory with your data file. Select the *tp.tp* file and click *Open*. If the executable or source files are not in the current directory, the Thread Profiler asks for the location of these files.

Binary instrumentation mode: Make sure to compile your program with debug information generation enabled (**-g**).

On Linux, you have to run your program under control of the Thread Profiler using **tprofile_cl**. After the program has finished, a short summary of the analysis is written to the console. The whole analysis data is written to the *threadprofiler* directory. Open the Thread Profiler Windows GUI as described above and select the *threadprofiler/tprofile.<pid>.tp* file.

Linux Example:

$ $PSRC/pex/832|| `tprofile_cl a.out`

To start the Thread Profiler GUI on Windows, click *Start* →*Programs* →*Intel Software Development Tools* →*Intel Thread Profiler* →*Intel Thread Profiler*. Then select *New Project* →*Threading Wizards* →*Intel Thread Profiler Wizard* →*OK*, choose an executable file, Collection Mode and finally click *Finish*.

8.4 Acumem ThreadSpotter (Lin)

Acumem ThreadSpotter[63] is a tool designed to automatically find memory-related performance problems in an application. The goal is to help non-expert programmers to optimize their code for multicore platforms. Acumem analyzes the execution of unmodified binaries and identifies 10 issue types. Possible solutions to found issues are presented. General feedback on data locality quality is given for many cache sizes evaluated from a single run.

It is not necessary to recompile your executable. We recommend, however, to compile with debugging information (-g option) in order to see the source of identified performance issues.

Use **module load acumem** to set up the environment. You can either use the GUI

$ `acumem`

to sample your program and generate a report or run these command lines:

$ $PSRC/pex/830|| `sample -o sample.smp -s 1000 -r a.out`

$ $PSRC/pex/830|| `report -o report -i sample.smp`

$ $PSRC/pex/830|| `firefox report.html`

The *sample* program instruments and runs the executable. During execution the sample.smp file is written. From this file the *report* program generates HTML files that can be viewed with a web browser, e.g. Firefox.

8.5 Frequency Analysis with tcov (Lin)

For error detection and tuning it might be helpful to know how often each statement is executed. For testing a program it is important that all program branches are passed (*test coverage*). A part of the Oracle Studio software called **tcov** can help in this test. The program must be compiled and linked with the Oracle compiler using the option **-xprofile=tcov**. In order to get reliable information on otherwise inlined functions, inlining should not be turned on. Optimization may cause inaccuracies as well.

In the following program execution, the frequencies of all statements are recorded. The values can be entered in modified program sources using the command

[63] Previous name of the Acumem tool was *Virtual Performance Expert*; the *SlowSpotter* is the marketing name of the single-threaded Acumem tool.

```
$ $PSRC/pex/840|| tcov -a -x a.out.profile pi.c
$ $PSRC/pex/840|| less pi.c.tcov
```

Statements which have never been executed are marked by "#####".
Note: To access **tcov**, the Oracle Studio compiler must be loaded.

8.6 Runtime Analysis with gprof (Lin)

With **gprof**, a runtime profile can be generated. The program must be translated and linked with the option **-pg**. During the execution a file named **gmon.out** is generated that can be analyzed by

```
$ gprof program
```

With **gprof** it is easy to find out the number of the calls of a program module, which is a useful information for inlining.
Note: *gprof* assumes that all calls of a module are equally expensive, which is not always true. We recommend using the Callers-Callees info in the Oracle Performance Analyzer to gather this kind of information as it is much more reliable. However, gprof is useful to get the *exact* function call counts.

8.7 Access to hardware counters using PAPI (Lin)

PAPI aims to provide the tool designer and application engineer with a consistent interface and methodology for use of the performance counter hardware found in most major microprocessors. While VTune, PTU and the Oracle Collector and Analyzer offer a GUI, PAPI is only a library. However, Vampir (see chapter 8.8.3 on page 79) and Scalasca (see chapter 8.8.4 on page 82) use PAPI as well and they come with a GUI. For more information on PAPI please refer to the webpage http://icl.cs.utk.edu/papi/

PAPI needs the **perfctr** (or other) Linux kernel patch. The number of measureable counters depends on the CPU and PAPI version. A couple of important counters are denoted in the table 8.22 on page 77.

Name	Code	derived	description
PAPI_L1_DCM	0x80000000	No	Level 1 data cache misses (n.a on Nehalem)
PAPI_L2_DCM	0x80000002	Yes	Level 2 data cache misses
PAPI_L1_TCM	0x80000006	No	Level 1 cache misses (n.a on Nehalem)
PAPI_L2_TCM	0x80000007	No	Level 2 cache misses
PAPI_TOT_IIS	0x80000031	No	Instructions issued
PAPI_TOT_INS	0x80000032	No	Instructions completed
PAPI_TOT_CYC	0x8000003b	No	Total cycles
PAPI_FP_OPS	0x80000066	Yes	Floating point operations

Table 8.22: A couple of available PAPI counters

By combining these counters you can obtain some basic metrics allowing you to evaluate your application, e.g.:

- **MFlops**: use PAPI_TOT_CYC and PAPI_FP_OPS
- **MIPs**: use PAPI_TOT_CYC and PAPI_TOT_INS

To use PAPI, load the appropiate modules

```
$ module load UNITE
$ module load papi
```

To get a list of available counters and information on how many counters you can use concurrently on the system, start

```
$ papi_avail
```

PAPI also supports native counters; a list can be obtained through

```
$ papi_native_avail
```

PAPI comes with several more tools. The **papi_event_chooser** command lets you test which counter can be combined, and the **papi_comand_line** utility program helps you to try out sets of counters quite effortlessly. The **papi_mem_info** command provides memory cache and TLB hierarchy information of the current processor. For further information on these and even more utilities please refer to the documentation at http://icl.cs.utk.edu/papi/docs/

8.8 Runtime Analysis of MPI Programs

8.8.1 Oracle Sampling Collector and Performance Analyzer (Lin)

With MPI programs, the Sampling Collector (see chapter 8.1.1 on page 71) can collect runtime information for each MPI process, which can also be displayed for each task separately. This technique is no longer supported to collect MPI trace data, but it can still be used for all other types of data.

```
$ mpiexec -np n collect a.out
```

With the **-m on** option of the Sampling Collector, MPI events can be traced as well.[64] Use the **-M** option to set the version of MPI to be used; selectable values are CT8.2, CT8.1, CT8, CT7, CT7.1, OPENMPI, MPICH2 or MVAPICH2. Use MPICH2 value for Intel MPI programs and the CTx.y value for Cluster Tools (Oracle MPI). The **--** (two minuses) flag has to be set before the executable file in order to allow **collect** to distinguish where the executable file is.

```
$ collect -m on -M CT8.2 mpiexec -np n -- a.out
```

The **-g** *experiment_group***.erg** option bundles experiments to an experiment group. The result of an experiment group can be displayed with the Analyzer

```
$ analyzer experiment_group
```

When collect is run with a large number of MPI processes, the amount of experiment data might become overwhelming. Try to start your program with as few processes as possible.

8.8.2 Intel Trace Analyzer and Collector (ITAC) (Lin / Win)

The Intel Trace Collector (ITC) for MPI applications produces event trace data. It is based on VampirTrace. The Intel Trace Analyzer (ITA) is a graphical tool that analyzes and displays the trace files generated by the ITC.

The tools help to understand the behavior of the application and to detect programming errors and performance problems. Please note that these tools are designed to be used with Intel or GNU compilers and Intel MPI.

On Linux, initialize the environment with

```
$ module load intelitac.
```

Profiling of dynamically linked binaries without recompilation: This mode is applicable to programs which use Intel MPI. In this mode, only MPI calls will be traced, which often suffices.

Run the binary under control of ITC by using the **-trace** command line argument of Intel MPI Library **mpiexec** wrapper. A message from the Trace Collector should appear that indicates where the collected information is saved (an **.stf** file). Use the ITA GUI to analyze this trace file. On Linux, start the Analyzer GUI with

```
$ traceanalyzer <somefile>.stf.
```

[64] MPI profiling is based on the open source VampirTrace 5.5.3 release.

Example:

```
$  $MPIEXEC -trace -np 2 a.out
...  skipped output ...
[0] Intel(R) Trace Collector INFO: Writing tracefile a.out.stf
```

`$  traceanalyzer a.out.stf` There also exists a Command Line Interface (CLI) of the Trace Analyzer on Linux. Please refer to the manual.

On Windows, start the Analyzer GUI by *Start →Programs →Intel Software Development Tools →Intel Trace Analyzer and Collector →Intel Trace Analyzer*, and open the trace file.

Trace files produced on Linux may be analyzed on Windows and vice versa.

Compiler-driven Subroutine Instrumentation allows you to trace the whole program additionally to the MPI library. In this mode the user-defined non-MPI functions are traced as well. Function tracing can easily generate huge amounts of trace data, especially for object-oriented programs. For the Intel compilers, use the flag **-tcollect** (on Linux) or **/Qtcollect** (on Windows) to enable the collecting. The switch accepts an optional argument to specify the collecting library to link. For example, for non-MPI applications you can select **libVTcs**: **-tcollect=VTcs**. The default value is **VT**.

Use the **-finstrument-function** flag with GNU Compilers to compile the object files that contain functions to be traced. ITC is then able to obtain information about the functions in the executable.

Run the compiled binary the usual way. After the program terminates, you get a message from the Trace Collector which says where the collected information is saved (an **.stf** file). This file can be analyzed with the ITA GUI in an usual way.

Linux Example:

```
$  $MPIEXEC -np 2 a.out
...  skipped output ...
[0] Intel(R) Trace Collector INFO: Writing tracefile a.out.stf
$  traceanalyzer a.out.stf
```

There are a lot of other features and operating modes, e.g. binary instrumentation with **itcpin**, tracing of non-correct programs (e.g. containing deadlocks), tracing MPI File IO and more.

More documentation on ITAC may be found in */opt/intel/itac/<VERSION>/doc* and on http://www.intel.com/cd/software/products/asmo-na/eng/cluster/tanalyzer/index.htm.

8.8.3 Vampir (Lin)

Vampir is a framework for the collection and visualization of event-based performance data. The collection of events is managed by a set of libraries that are activated at link time. It consists of two separate units, the instrumentation and measurement package *vampirtrace* and the visualization package *vampir* or *vampir next generation*. This tool is currently deployed in collaboration with the VI-HPS group.

Measurement: Vampir is a tool suitable for the analysis of parallel and distributed applications and allows the tracing of MPI communication as well as OpenMP events. Additionally, certain program-specific events and data from hardware event counters can also be measured. Vampir is designed to help you to find performance bottlenecks in your application. Such bottlenecks originate from computation-, communication-, memory-, and I/O-aspects of your application in conjunction with the hardware setup.
Note: Measurement may significantly disturb the runtime behavior of your application.

Possible bottlenecks identifiable through the use of VampirTrace are:

- Unbalanced computation

- Strictly serial parts of your program
- Very frequent tiny function calls
- Sparse loops
- Communication dominating over computation
- Late sender, late receiver
- Point-to-point messages instead of collective communication
- Unmatched messages
- Overcharge of MPI's buffers
- Bursts of large messages
- Frequent short messages
- Unnecessary synchronization
- Memory-bound computation (detectable via hardware event counters)
- I/O-bound computation (slow input/output, sequential I/O on single process, I/O load imbalance)

Be aware that tracing can cause substantial additional overhead and may produce lots of data, which will ultimately perturb your application runtime behavior during measurement.

To be able to spot potential bottlenecks, the traces created with VampirTrace are visualized with either Vampir or VampirServer. These GUIs offer a large selection of views, like global timeline, process timeline, counter display, summary chart, summary timeline, message statistics, collective communication statistics, counter timeline, I/O event display and call tree (compare figure 8.1 on page 81).

Setup: Before you start using Vampir, the appropriate environment has to be set up. All Vampir modules only become accessible after loading the UNITE module:
`$ module load UNITE`
To do some tracing, you have to load the vampirtrace module:
`$ module load vampirtrace`
Later, once you have traced data that you want to analyze use:
`$ module load vampir`
to load the visualization package *vampir*.
Alternatively you have the choice to load *vampir next generation*:
`$ module load vampirserver`

Instrumentation: To perform automatic instrumentation of serial or OpenMPI codes, simply replace your compiler command with the appropriate vampir trace wrapper, for example:

`$CC` → `vtcc` `$CXX` → `vtcxx` `$FC` → `vtf90`
If your application uses MPI, you have to specify the MPI-compiler-wrapper for *vampirtrace* to ensure correct linking of the MPI libraries. For this, the option `-vt:cc`, `-vt:cxx` and `-vt:fc` is used for C, C++ and FORTRAN respectively.

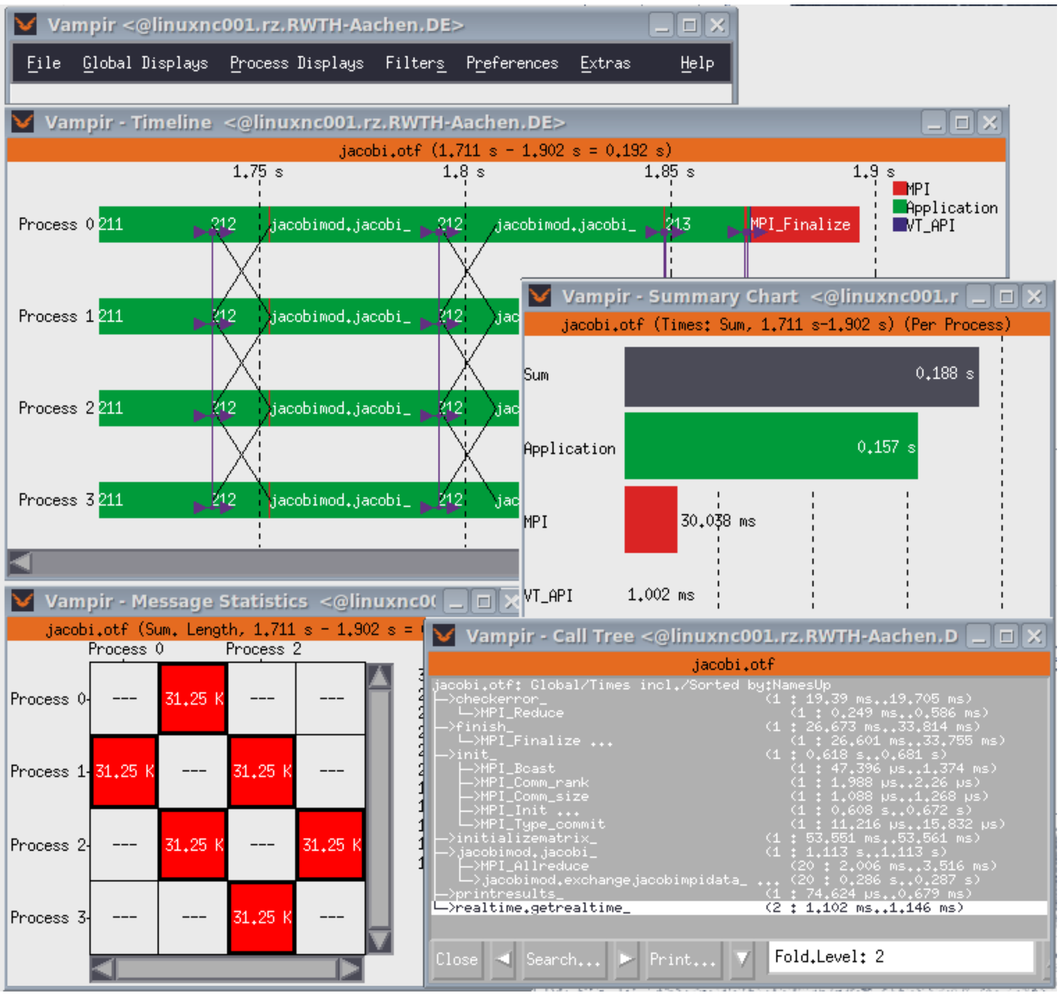


Figure 8.1: The Vampir GUI

Execution: Such an instrumented binary can then be executed as usually and will generate trace data during its execution. There are several environment variables to control the behavior of the measurement facility within the binary. Please refer to the *vampirtrace* documentation at http://tu-dresden.de/die_tu_dresden/zentrale_einrichtungen/zih/forschung/software_werkzeuge_zur_unterstuetzung_von_programmierung_und_optimierung/vampirtrace/dateien/VT-UserManual-5.8.2.pdf for more details.

Visualization: To start the analysis of your trace data with the classic *Vampir*, load the module, then simply type:
`$ vampir` *`tracefilename.otf`*

To analyze with the more advanced (and multiprocessing) *vampir next generation* the server needs to be started, if not already running, prior to analysis. Assuming the module environment has been set up properly, calling
`$ vngd-start.sh`
starts this server and will (after possibly a few seconds) return a line similar to:
`Server listens on: linuxscc005.rz.RWTH-Aachen.DE` 33071.
The server is now ready and waits for a connection on linuxscc005 at port 33071. To connect to this server, start a new console, load the *vampir* module as described above and connect to the server through:
File->Remote_Open->enter:servername and port->Insert/Update->Connect->select: Path of

trace->Open

Both ways will start the Vampir GUI.

You can also watch the flash-tutorials offered by http://www.vampir.eu at
http://www.vampir.eu/flash/tutorial/Vampir Tutorial/tutorial.html.

Example in C, summing up all three steps:
$ $PSRC/pex/860|| `vtcc -vt:cc $MPICC $FLAGS_DEBUG $PSRC/cmj/*.c`
$ $PSRC/pex/860|| `$MPIEXEC -np 4 a.out`
$ $PSRC/pex/860|| `vampir a.otf`

Note: Vampir displays information for each process, therefore the GUI will be crowded with more than about 16 processes and analysis may be not possible.

8.8.4 Scalasca (Lin)

Scalasca, similar to Vampir, is a performance analysis tool suite. Scalasca is designed to automatically spot typical performance problems in parallel running applications with large counts of processes or threads. Scalasca displays a large number of metrics in a tree view, describing your application run. Scalasca presents different classes of metrics to you: generic, MPI-related and OpenMP-related ones.
Generic metrics:

- Total CPU allocation time
- Execution time without overhead
- Time spent in tasks related to measurement (does not include per-function perturbation!)
- Number of times a function/region was executed
- Aggregated counter values for each function/region

MPI-related metrics:

- Total CPU allocation time
- Time spent in pre-instrumented MPI functions
- Time spent in MPI communication calls, subdivided into collective and point-to-point
- Time spent in calls to MPI_Barrier
- Time spent in MPI I/O functions
- Time spent in MPI_Init and MPI_Finalize

OpenMP-related metrics:

- Total CPU allocation time
- Time spent for OpenMP-related tasks
- Time spent for synchronizing OpenMP threads
- Time spent by master thread to create thread teams
- Time spent in OpenMP flush directives
- Time spent idle on CPUs reserved for slave threads

Setup: Use
`$ module load UNITE; module load scalasca`
to load the current default version of scalasca.

Instrumentation: To perform automatic instrumentation of serial or MPI codes, simply put the command for the scalasca wrapper in front of your compiler and linker commands. For OpenMP codes the additional flag `-pomp` is necessary. For example:

`gcc` → `skin gcc` or `ifort` → `skin -pomp ifort`

Execution: To execute such an instrumented binary, prepend `scan` to your normal launch-line. This will properly set up the measurement environment and analyze data during the program execution. There are two possible modes that you can use with Scalasca.

- A faster but less detailed **profile mode** is selected by `scan -p` (default), which gathers statistical data of your application, like function visits and percentage of total runtime. After execution there will be a directory called `epik_<YourApplicationName>` in your working directory containing the results of the analysis run.

- The second mode `scan -t` will trigger the more detailed **tracing mode**, which will gather very detailed information. This will almost certainly increase your execution time by a substantial amount (up to a factor of 500 for function call intensive and template codes). In this tracing mode Scalasca automatically performs a parallel analysis after your application's execution. As with profiling there will be a new directory containing the data with the name of `epik_<YourApplicationName>_<NumberOfProcesses>_trace`.
  `$ scan -t $MPIEXEC -np 4 a.out` will start the executable *a.out* with four processes and will trace its behavior generating a data directory `epik_a_4_trace`. There are several environment options to control the behavior of the measurement facility within the binary.

Note: Existing measurement directories will not be overwritten and will block program execution.

Visualization: To start analysis of your trace data call `square` *`scalascaDataDirectory`* where *scalascaDataDirectory* is the directory created during your program execution. This will bring up the cube3 GUI and display performance data about your application. Please refer to the *scalasca* http://www.scalasca.org/software/documentation for more details.

Example in C, summing up all three steps:

```
$ $PSRC/pex/870|| skin $MPICC $FLAGS_DEBUG $FLAGS_FAST $FLAGS_ARCH64 $PSRC/cmj/*.c
$ $PSRC/pex/870|| scan $MPIEXEC "-show -x EPK_TRACE -x EPK_TITLE -x EPK_LDIR -x EPK_GDIR -x ELG_BUFFER_SIZE" -np 4 a.out
$ $PSRC/pex/870|| square epik_a_4_sum
```

Note: Instead of `skin`, `scan` and `square`, you can also use `scalasca -instrument`, `scalasca -analyse` and `scalasca -examine`.

8.8.5 MARMOT (Lin)

MARMOT is a tool to check MPI programs for MPI problems and correctness. During runtime it surveys the MPI-calls made and automatically checks the correct usage of these calls and their arguments. The observations are saved for later post-mortem analysis. MARMOT is a library and has to be linked to your application along with the MPI library. It currently supports C/Fortran language binding of MPI-1.2 and also works with hybrid applications using MPI and OpenMP. MARMOT detects the incorrect usage of MPI, non-portable constructs, possible race-conditions and deadlocks. The results can be presented in ASCII text, as a file for Cube (Scalasca's visualizer), or as HTML pages, which is the default on our cluster. See the web page http://www.hlrs.de/organization/av/amt/research/marmot/ to learn more about MARMOT.

Setup: Use
`$ module load UNITE; module load marmot`
to load the current default version of MARMOT.

Compilation: To compile your application with MARMOT, just replace the call to the compiler with the corresponding wrapper script supplied by MARMOT. For C code choose marmotcc, for C++ code choose marmotcxx and for FORTRAN code use marmotf90.
Example in c:

$ $PSRC/pex/880|| `marmotcc $FLAGS_DEBUG -o pi.exe $PSRC/C-mpi-pi/pi.c -lm`

If you use the *-show* flag you can see what the wrapper is doing:

`$ marmotcc -show $FLAGS_DEBUG -o pi.exe $PSRC/C-mpi-pi/pi.c -lm`

Exemplary output:

```
$ marmotcc -show -g -o pi.exe ./pi.c -lm icc -I/usr/local_rwth/sw/vihps/marmot/
2.2.1beta/intel_64_10.1.017-intel_mpi_3.1.038/include -g -o pi.exe ./pi.c
-lm -L/usr/local_rwth/sw/vihps/marmot/2.2.1beta/intel_64_10.1.017-intel
_mpi_3.1.038/lib -lmarmot-profile -lmarmot-core -L/opt/intel/impi/3.1.038/lib64
-lmpi -lpthread -L/usr/lib -lxml2 -L/rwthfs/rz/SW/VIHPS/UNITE/packages/scalasca/
1.1-intelmpi_3.2-intel_11.0-64/lib -lcube3 -lsz -L/usr/lib -lstdc++
-I/opt/intel/impi/3.1.038/include64 -L/opt/intel/impi/3.1.038/lib64 -Xlinker
-rpath -Xlinker $libdir -Xlinker -rpath -Xlinker /opt/intel/mpi-rt/3.1
-lmpi_dbg -lmpiif -lmpigi -lrt -lpthread -ldl
```

The exemplary output shows among other things that MARMOT is linked before Intel MPI. The MARMOT wrapper should always automatically include the needed MPI library, the OpenMP flag, if applicable, as well as the include paths and libraries. If you want to manually compile and link your application with MARMOT please refer to the user guide, which you can find at $MARMOT_ROOT/share/doc/marmot-*X.Y.Z*/marmot_userguide.pdf if you have already loaded the marmot module version *X.Y.Z*.

Execution: MARMOT requires an additional process. You therefore have to ask for one extra process.
Example in c:
$ $PSRC/pex/880|| `$MPIEXEC -np 5 pi.exe`

The example asks for 5 processes, 4 of which will go to the application and the remaining one is needed by the MARMOT debug server. On our clusters, the result will be presented as an HTML page.

To change this, the environment variable MARMOT_LOGFILE_TYPE can be set to 0 for ASCII Logging or 2 for CUBE Logging. To come back to our default, set it to 1. There are some more environment variables which can again be looked up in the MARMOT user guide. *Note:* Because of the used MPI-Wrapper of the cluster it may be necessary to pass the environment variable with *-x MARMOT_LOGFILE_TYPE*.

Figure 8.2 on page 85 shows a fragment from the HTML output file which is the result of this example:

MARMOT Logfile, date: Thu Apr 2 09:54:04 2009.

Timestamp	Rank	Thread	Type	Message	Location	MPI-Standard Reference
0	Global	0	**Information**	Text: The following entries describe all the Environmental variables.	Unknown	
0	Global	0	**Information**	Text: MARMOT_DEBUG_MODE = 2 (negative int: nothing, 0: errors, 1: errors and warnings, 2: errors, warnings and remarks are reported, default: 2)	Unknown	
5	Global	0	**Warning**	Text: Processes 0 and 1 both run on linuxscc002	Unknown	Infos see MPI-Standard
5	Global	0	**Warning**	Text: Processes 0 and 2 both run on linuxscc002	Unknown	Infos see MPI-Standard
5	Global	0	**Warning**	Text: Processes 1 and 2 both run on linuxscc002	Unknown	Infos see MPI-Standard
17	1	0	**Note**	Text: Note: The minimal threadlevel required by this run was: MPI_THREAD_SINGLE Call: MPI_Finalize	./pi.c line: 57	
18	2	0	**Note**	Text: Note: The minimal threadlevel required by this run was: MPI_THREAD_SINGLE Call: MPI_Finalize	./pi.c line: 57	
18	0	0	**Note**	Text: Note: The minimal threadlevel required by this run was: MPI_THREAD_SINGLE Call: MPI_Finalize	./pi.c line: 57	

Figure 8.2: Marmot HTML Output

9 Application Software and Program Libraries

9.1 Application Software

You can find a list of available application software and program libraries form several ISVs at http://www.rz.rwth-aachen.de/go/id/ond/lang/en/

As for the compiler and MPI suites, we also offer environment variables for the mathematical libraries to make usage and switching easier. These are **FLAGS_MATH_INCLUDE** for the include options and **FLAGS_MATH_LINKER** for linking the libraries. If loading more than one mathematical module, the last loaded will overwrite and/or modify these variables. However (almost) each module sets extra variables that will not be overwritten.

9.2 BLAS, LAPACK, BLACS, ScaLAPACK, FFT and other libraries

If you want to use BLAS, LAPACK, BLACS, ScaLAPACK or FFT you are encouraged to read the chapters about optimized libraries: Intel MKL (recommended, see 9.3 on page 85), Oracle (Sun) Performance Library (see 9.4 on page 86), ACML (see 9.5 on page 87). The optimized libraries usually provide very good performance and do not only include the above-mentioned but also some other libraries.

Alternatively, you are free to use the native Netlib implementations - just download the source and install the libraries in your home. Note: The self-compiled versions from Netlib usually provide much lower performance than the optimized versions.

9.3 MKL - Intel Math Kernel Library (Lin / Win)

The Intel Math Kernel Library (Intel MKL) is a library of highly optimized, extensively threaded math routines for science, engineering, and financial applications. This library is optimized for Intel processors, but it works on AMD Opteron machines as well.

Intel MKL contains an implementation of BLAS, BLACS, LAPACK and ScaLAPACK, Fast Fourier Transforms, Sparse Solvers (Direct - PARDISO, Iterative - FGMRES and Conjugate Gradient Solvers), Vector Math Library and Vector Random Number Generators.

Starting with version 11 of Intel compiler, a version of MKL is included in the compiler distribution and the environment is initialized if the compiler is loaded.

If you want to use an alternative version of MKL with a given Intel 11 compiler, you have to initialize the environment of this MKL version *after* the compiler. To use Intel MKL with another compiler, load this compiler at last and then load the MKL environment.

To initialize the Intel MKL environment, use
`$ module load LIBRARIES; module load intelmkl`.

This will set the environment variables **FLAGS_MKL_INCLUDE** and **FLAGS_MKL_LINKER** for compiling and linking, which are the same as the **FLAGS_MATH_..** if the MKL module was loaded last. These variables let you use at least the BLAS and LAPACK routines of Intel MKL. To use other capabilities of Intel MKL, please refer to the Intel MKL documentation http://www.intel.com/cd/software/products/asmo-na/eng/345631.htm

The BLACS and ScaLAPACK routines use Intel MPI, so you have to load the Intel MPI before compiling and running a program which uses BLASCS or ScaLAPACK.

The Intel MKL contains a couple of OpenMP parallelized routines, and up to version 10.0.3.020 runs in parallel by default if it is called from a non-threaded program. Be aware of this behavior and disable parallelism of the MKL if needed. The number of threads the MKL uses is set by the environment variable **OMP_NUM_THREADS** or **MKL_NUM_THREADS**.

There are two possibilties for calling the MKL routines from C/C++.

1. **Using BLAS**
   You can use the Fortran-style routines directly. Please follow the Fortran-style calling conventions (call-by-reference, column-major order of data). Example:
   ```
   $ $PSRC/pex/950|| $CC $FLAGS_MATH_INCLUDE -c $PSRC/psr/useblas.c
   $ $PSRC/pex/950|| $FC $FLAGS_MATH_LINKER $PSRC/psr/useblas.o
   ```

2. **Using CBLAS**
   Using the BLAS routines with the C-style interface is the preferred way because you don't need to know the exact differences between C and Fortran and the compiler is able to report errors before runtime. Example:
   ```
   $ $PSRC/pex/950.1|| $CC $FLAGS_MATH_INCLUDE -c $PSRC/psr/usecblas.c
   $ $PSRC/pex/950.1|| $CC $FLAGS_MATH_LINKER $PSRC/psr/usecblas.o
   ```

Please refer to Chapter 7 in the Intel MKL User's Guide[65] for details with mixed language programming.

9.4 The Oracle (Sun) Performance Library (Lin)

The Oracle (Sun) Performance Library is part of the Oracle Studio software and contains highly optimized and parallelized versions of the well-known standard public domain libraries available from Netlib http://www.netlib.org: LAPACK version 3, BLAS, FFTPACK version 4 and VFFTPACK version 2.1 from the field of linear algebra, Fast Fourier transforms and solution of sparse linear systems of equations (Sparse Solver **SuperLU**, see http://crd.lbl.gov/~xiaoye/SuperLU/).

The studio module sets the necessary environment variables.

To use the Oracle performance library link your program with the compiler option **-xlic_lib=sunperf**. The performance of FORTRAN programs using the BLAS-library and/or intrinsic functions can be improved with the compiler option **-xknown_lib=blas,intrinsics**. The corresponding routines will be inlined if possible.

[65] http://www.intel.com/software/products/mkl/docs/linux/webhelp/userguide.htm

The Performance Library contains parallelized sparse BLAS routines for matrix-matrix multiplication and a sparse triangular solver. Linpack routines are no longer provided. It is strongly recommended to use the corresponding LAPACK routines instead.

Many of the contained routines have been parallelized using the shared memory programming model. Compare the execution times! To use multiple threads set the **OMP_NUM_THREADS** variable accordingly.

```
$ $PSRC/pex/920|| export OMP_NUM_THREADS=4;
$ $PSRC/pex/920|| $CC $FLAGS_MATH_INCLUDE $FLAGS_MATH_LINKER $PSRC/psr/useblas.c
```

The number of threads used by the parallel Oracle Performance Library can also be controlled by a call to its **use_threads(n)** function, which overrides the OMP_NUM_THREADS value.

Nested parallelism is not supported; Oracle Performance Library calls made from a parallel region will not be further parallelized.

9.5 ACML - AMD Core Math Library (Lin)

The AMD Core Math Library (ACML) incorporates BLAS, LAPACK and FFT routines that are designed for performance on AMD platforms, but the ACML works on Intel processors as well. There are OpenMP parallelized versions of this library, are recognizable by an *_mt* appended to the version string. If you use the OpenMP version don't forget to use the OpenMP flags of the compiler while linking.

To initialize the environment, use

`$ module load LIBRARIES; module load acml`. This will set the environment variables **FLAGS_ACML_INCLUDE** and **FLAGS_ACML_LINKER** for compiling and linking, which are the same as the **FLAGS_MATH_..** if the ACML module was loaded last.

Example:

```
$ $PSRC/pex/941|| $CC $FLAGS_MATH_INCLUDE -c $PSRC/psr/useblas.c
$ $PSRC/pex/941|| $FC $FLAGS_MATH_LINKER $PSRC/psr/useblas.o
```

9.6 Nag Numerical Libraries (Lin)

The Nag Numerical Libraries provide a broad range of reliable and robust numerical and statistical routines in areas such as optimization, PDEs, ODEs, FFTs, correlation and regression, and multivariate methods, to name just a few.

The following NAG Numerical Components are available:

1. NAG **C** Library: A collection of over 1,000 algorithms for mathematical and statistical computation for C/C++ programmers. Written in C, these routines can be accessed from other languages, including C++ and Java.

2. NAG **Fortran** Library: A collection of over 1,600 routines for mathematical and statistical computation. This library remains at the core of NAG's product portfolio. Written in Fortran, the algorithms are usable from a wide range of languages and packages including Java, MATLAB, .NET/C# and many more.

3. NAG **Fortran 90** Library: A collection of over 200 generic user-callable procedures, giving easy access to complex and highly sophisticated algorithms each designed and implemented using the performance, simplicity and flexibility of Fortran 90/95. These are equivalent to well over 440 routines in the NAG Fortran Library.

4. NAG **SMP** Library: A numerical library containing over 220 routines that have been optimized or enhanced for use on Symmetric Multi-Processor (SMP) computers. The NAG SMP Library also includes the full functionality of the NAG Fortran Library. It

is easy to use and link due to identical interface to the NAG FORTRAN Library. On his part, the NAG SMP library uses routines from the BLAS/LAPACK library.

5. NAG **Parallel** Library: A high-performance computing library consisting of 180 routines that have been developed for distributed memory systems. The interfaces have been designed to be as close as possible to equivalent routines in the NAG FORTRAN Library. The components of the NAG Parallel Library hide the message passing (MPI) details in underlying tiers (BLACS, ScaLAPACK).

To use the NAG components you have to load the LIBRARIES module environment first:
`$ module load LIBRARIES`
To find out which versions of NAG libraries are available, use
`$ module avail nag`
To set up your environment for the appropriate version, use the **module load** command, e.g. for the NAG FORTRAN library (Mk22):
`$ module load nag/fortran_mark22`
This will set the environment variables **FLAGS_MATH_INCLUDE**, **FLAGS_MATH_LINKER** and also **FLAGS_NAG_INCLUDE**, **FLAGS_NAG_LINKER**.

Example:
`$ $PSRC/pex/970|| $FC $FLAGS_MATH_INCLUDE $FLAGS_MATH_LINKER $PSRC/psr/usenag.f`

Note: All above mentioned libraries are installed as 64bit versions.
Note: For **FORTRAN**, **FORTRAN 90** and **c** libraries both FLAGS_MATH_... or FLAGS_NAG_.... environment variables can be used.
Note: The **FORTRAN 90** libraries are available for Intel and Oracle Studio compilers only.
Note: The **smp** library needs an implementation of a BLAS/LAPACK library. If using Intel compiler, the enclosed implementation of Intel MKL will be used automatically if you use the **FLAGS_MATH_INCLUDE** and **FLAGS_MATH_LINKER** flags. The **FLAGS_NAG_INCLUDE** and **FLAGS_NAG_LINKER** variables provide a possibility of using NAG **smp** with other compilers and BLAS/LAPACK implementations. *Note*: The **parallel** library needs an implementation of a BLACS/ScaLAPACK and those need a MPI library. If using the Intel compiler, the enclosed implementation of Intel MKL will be used automatically to provide BLACS/ScaLAPACK if you use the **FLAGS_MATH_INCLUDE** and **FLAGS_MATH_LINKER** flags. However, the MKL implementation of BLACS/ScaLAPACK is known to run with Intel MPI only, so you have to switch your MPI by typing **module switch sunmpi intelmpi** *before* loading the NAG **parallel** library. The usage of any another compiler and/or BLACS/ScaLAPACK library with the NAG **parallel** library is in principle possible but not supported through the modules now.
Would You Like To Know More? http://www.nag.co.uk/numeric/numerical_libraries.asp

9.7 TBB - Intel Threading Building Blocks (Lin / Win)

Intel Threading Building Blocks is a runtime-based threaded parallel programming model for C++ code. It consists of a template-based runtime library to help you to use the performance of multicore processors. More information can be found at http://www.threadingbuildingblocks.org/.

On Linux, a release of TBB is included into Intel compiler releases and thus no additional module needs to be loaded. Additionally there are alternative releases which may be initialized by loading the corresponding modules:
`$ module load inteltbb`
Use the environment variables $LIBRARY_PATH and $CPATH for compiling and linking. To link TBB set the **-ltbb** flag. With **-ltbb_debug** you may link a version of TBB which provides some debug help.

Linux Example:
$ $PSRC/pex/961|| $CXX -O2 -DNDEBUG -I$CPATH -o ParallelSum ParSum.cpp -ltbb
$ $PSRC/pex/961|| ./ParSum

Use the debug version of TBB:
$ $PSRC/pex/962|| $CXX -O0 -g -DTBB_DO_ASSERT $CXXFLAGS -I$CPATH -o ParSum_debug ParallelSum.cpp -ltbb_debug
$ $PSRC/pex/962|| ./ParSum_debug

On Windows, the approach is the same, i.e. you have to link with the TBB library and set the library and include path. The Intel TBB installation is located in *C:\Program Files (x86)\Intel\TBB\<VERSION>*.

Select the appropriate version of the library according to your environment:

- **em64t** or **ia32** (for 64bit or 32bit programs)
- **vc8** (Visual Studio 2005) or **vc9** (Visual Studio 2008)

9.8 R_Lib (Lin)

The r_lib is a Library that provides useful functions for time measurement, processor binding and memory migration, among other things. It can be used under Linux. An r_lib library version for Windows is under development.

Example:
$ $PSRC/pex/960|| $CC -L/usr/local_rwth/lib64 -L/usr/local_rwth/lib -lr_lib -I/usr/local_rwth/include $PSRC/psr/rlib.c

The following sections describe the available functions for C/C++ and Fortran.

9.8.1 Timing

double r_ctime(void) - returns user and system CPU time of the running process and its children in seconds
double r_rtime(void) - returns the elapsed wall clock time in seconds
char* r_time(void) - returns the current time in the format hh:mm:ss
char* r_date(void) - returns the current date in the format yy.mm.dd

Example in C

```
#include "r_lib.h"
/* Real and CPU time in seconds as double */
double realtime, cputime;
realtime = r_rtime();
cputime = r_ctime();
```

and in Fortran

```
! Real and CPU time in seconds
REAL*8 realtime, cputime, r_rtime, r_ctime
realtime = r_rtime()
cputime = r_ctime()
```

Users' CPU time measurements have a lower precision and are more time-consuming. In case of parallel programs, real-time measurements should be preferred anyway!

9.8.2 Processor Binding

The following calls automatically bind processes or threads to empty processors.
void r_processorbind(int p) - binds current thread to a specific CPU
void r_mpi_processorbind(void) - binds all MPI processes
void r_omp_processorbind(void) - binds all OpenMP threads
void r_ompi_processorbind(void) - binds all threads of all MPI processes

Print out current bindings:
void r_mpi_processorprint(int iflag)
void r_omp_processorprint(int iflag)
void r_ompi_processorprint(int iflag)

9.8.3 Memory Migration

int r_movepages(caddr_t addr, size_t len) - Moves data to the processor where the calling process/thread is running. *addr* is the start address and *len* the length of the data to be moved in byte.
int r_madvise(caddr_t addr, size_t len, int advice) - If the advise equals 7, the specified data is moved to the thread that uses it next.

9.8.4 Other Functions

char* r_getenv(char* envnam) - Gets the value of an environment variable.
int r_gethostname(char *hostname, int len) - Returns the hostname.
int r_getcpuid(void) - Returns processor ID.
void r_system(char *cmd) - Executes a shell command.
Details are described in the manual page (**man r_lib**). If you are interested in the r_lib sources please contact us.

10 Miscellaneous

10.1 Useful Commands (Lin)

csplit	Splits C programs
fsplit[67]	Splits FORTRAN programs
nm	Prints the name list of object programs
ldd	Prints the dynamic dependencies of executable programs
ld	Runtime linker for dynamic objects
readelf	Displays information about ELF format object files.
vmstat	Status of the virtual memory organization
iostat	I/O statistics
sar	Collects, reports, or saves system activity information
mpstat	Reports processor related statistics
lint[67]	More accurate syntax examination of C programs
dumpstabs[67]	Analysis of an object program (included in Oracle Studio)
pstack **pmap**	Analysis of the /proc directory
cat /proc/cpuinfo	Processor information
free	Shows how much memory is used
top	Process list
strace	Logs system calls
file	Determines file type
uname -a	Prints name of current system
ulimit -a	Sets/gets limitations on the system resources
which *command*	Shows the full path of *command*
dos2unix, unix2dos	DOS to UNIX text file format converter and vice versa
screen	Full-screen window manager that multiplexes a physical terminal

10.2 Useful Commands (Win)

hostname	Prints name of current system
quota	Shows quota values for Home and Work
set	Prints environment variables
where *cmmd*	Shows full path of the *cmmd* command
windiff	Compares files/directories (graphically)

[67] *Note*: The utilities **fsplit**, **lint**, **dumpstabs** are shipped with Oracle Studio compilers, thus you have to load the **studio** module to use them:

```
$ module load studio
```

A Debugging with TotalView - Quick Reference Guide (Lin)

This quick reference guide describes how to debug serial and parallel (OpenMP and MPI) programs written in C, C++ or FORTRAN 90/95, using the TotalView debugger from TotalView Technologies on the RWTH Aachen HPC-Cluster, in a very condensed form.

For further information about TotalView see http://www.totalviewtech.com. Especially the User's Manual[68] and the Reference Guide[69] could provide a lot useful information about TotalView.

A.1 Debugging Serial Programs

A.1.1 Some General Hints for Using TotalView

- Click your middle mouse button to **dive** on things in order to get more information.
- Return (**undive**) by clicking on the *undive* button (if available), or by *View* →*Undive*.
- You can change all highlighted values (Press F2).
- If at any time the source pane of the process window shows disassembled machine code, the program was stopped in some internal routine. Select the first user routine in the **Stack Trace Pane** in order to see where this internal routine was invoked.

A.1.2 Compiling and Linking

Before debugging, compile your program with the option **-g** and without any optimization.

A.1.3 Starting TotalView

You can debug your program

1. either by starting TotalView with your program as a parameter
   `$ $PSRC/pex/a10|| totalview a.out [ -a options ]`
2. or by starting your program first and then attaching TotalView to it. In this case start
   `$ totalview`

   which first opens its *New Program* dialog. This dialog allows you to choose the program you want to debug.
3. You can also analyze the core dump after your program crashed by
   `$ totalview a.out core`

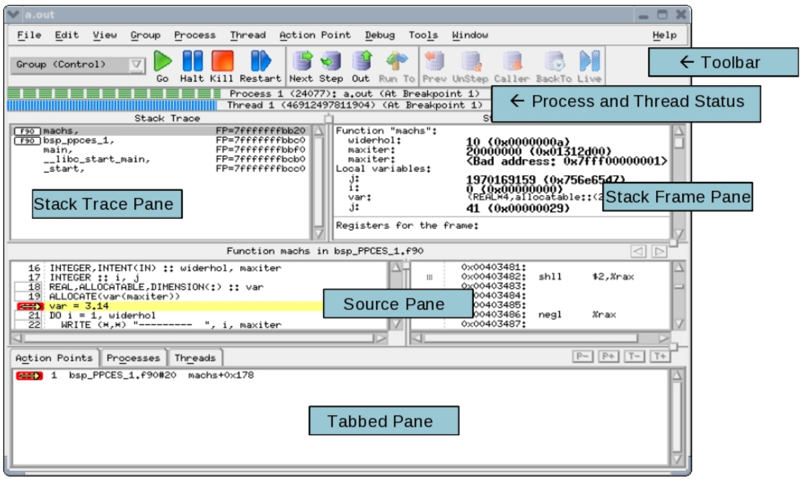
Start Parameters (runtime arguments, environment variables, standard IO) can be set in the *Process* →*Startup Parameters ...* menu.

After starting your program, TotalView opens the *Process Window*. It consists of

- the *Source Pane*, displaying your program's source code;
- the *Stack Trace Pane*, displaying the call stack;
- the *Stack Frame Pane*, displaying all the variables associated with the selected stack routine;
- the *Tabbed Pane*, showing the threads of the current process (*Threads* subpane), the MPI processes (*Processes* subpane), and listing all breakpoints, action points and evaluation points (*Action Points Threads* subpane);

[68] http://www.totalviewtech.com/pdf/other/TotalView_User_Guide.pdf
[69] http://www.totalviewtech.com/pdf/other/totalview_reference_guide.pdf

- the *Status Bar*, displaying the status of current process and thread;
- the *Toolbar*, containing the action buttons.



A.1.4 Setting a Breakpoint

- If the right function is already displayed in the *Source Pane*, just click on a boxed line number of an executable statement once to *set a breakpoint*. Clicking again will *delete the breakpoint*.
- Search the function with the *View →Lookup Function* command first.
- If the function is in the current call stack, dive on its name in the *Stack Trace Pane* first.
- Select *Action Points →At Location* and enter the function's name.

A.1.5 Starting, Stopping and Restarting your Program

- Start your program by selecting *Go* on the icon bar and stop it by selecting *Halt*.
- Set a *breakpoint* and select *Go* to run the program until it reaches the line containing the breakpoint.
- Select a program line and click on *Run To* on the icon bar.
- Step through a program line by line with the *Step* and *Next* commands. *Step* steps into and *Next* jumps over function calls.
- Leave the current function with the *Out* command.
- To restart a program, select *Restart*.

A.1.6 Printing a Variable

- The values of simple actual variables are displayed in the *Stack Frame Pane* of the *Process Window*.
- You may use the *View →Lookup Variable* command.
- When you dive (middle click) on a variable, a separate *Variable Window* will be opened.
- You can change the variable type in the *Variable Window* (*type casting*).

- If you are displaying an array, the *Slice* and *Filter* fields let you select which subset of the array will be shown (examples: Slice: (3:5,1:10:2), Filter: > 30).

- One and two-dimensional arrays or array slices can be graphically displayed by selecting *Tools* →*Visualize* in the *Variable Window*.

- If you are displaying a structure, you can look at substructures by rediving or by selecting *Dive* after clicking on the right mouse button.

A.1.7 Action Points: Breakpoints, Evaluation Points, Watchpoints

- The program will stop when it hits a *breakpoint*.

- You can temporarily introduce some additional C or FORTRAN style program lines at an *Evaluation Point*. After creating a breakpoint, right-click on the *STOP* sign and select *Properties* →*Evaluate* to type in your new program lines. Examples are shown in table A.23 on page 94.

An additional print statement: (FORTRAN write is not accepted)	**printf ("x = %f\n", x/20)**
Conditional breakpoint:	**if (i == 20) $stop**
Stop after every 20 executions:	**$count 20**
Jump to program line 78:	**goto $78**
Visualize an array	**$visualize a**

Table A.23: Action point examples

- A *watchpoint* monitors the value of a variable. Whenever the content of this variable (memory location) changes, the program stops. To set a watchpoint, dive on the variable to display its *Variable Window* and select the *Tools* →*Watchpoint* command.

You can save / reload your action points by selecting *Action Point* →*Save All* resp. *Load All*.

A.1.8 Memory Debugging

TotalView offers different memory debugging features. You can guard dynamically allocated memory so that the program stops if it violates the boundaries of an allocated block. You can hoard the memory so that the program will keep running when you try to access an already freed memory block. Painting the memory will cause errors more probably; especially reading and using uninitialized memory will produce errors. Furthermore you can detect memory leaks.

- Enable the memory debugging tool before you start your program by selecting the *Debug* entry from the tools menu and click the *Enable memory debugging* button.

- Set a breakpoint at any line and run your program into it.

- Open the Memory Debugging Window: select *Debug* →*Open MemoryScape*.

- Select the *Memory Reports* →*Leak Detection* tab and choose *Source report* or *Backtrace report*. You will then be presented with a list of Memory blocks that are leaking.

A.1.9 ReplayEngine

TotalView provides the possibility of reversely debugging your code by recording the execution history. The ReplayEngine restores the whole program states, which allows the developer to work back from a failure, error or even a crash. The ability of stepping forward and backward through your code can be very helpful and reduce the amount of time for debugging dramatically because you do not need to restart your application if you want to explore a previous program state. Furthermore the following replay items are supported:[70]

- Heap memory usage
- Process file and network I/O
- Thread context switches
- Multi-threaded applications
- MPI parallel applications
- Distributed applications
- Network applications

The following functionality is provided:

- First you need to activate the ReplayEngine: *Debug* →*Enable ReplayEngine*
- *GoBack* runs the program backwards to a previous *breakpoint*.
- *Prev* jumps backwards to the previous line (function call).
- *Unstep* steps backwards to the previous instruction within the function.
- *Caller* jumps backwards to the caller of the function.

A.1.10 Offline Debugging - TVScript

If interactive debugging is impossible, e.g. because the program has to be run in the batch system due to problem size, an interesting feature of the TotalView debugger called *TVScript* can be helpful. Use the **tvscript** shell command to define points of interest in your program and corresponding actions for TotalView to take. TVScript supports serial, multithreaded and MPI programming models and has full access to the memory debugging capabilities of TotalView. More information about TVScript can be found in Chapter 4 of the Reference Guide[71].

A.2 Debugging Parallel Programs

A.2.1 Some General Hints for Parallel Debugging

- If possible, make sure that your serial program runs fine first.
- Debugging a parallel program is not always easy. Use as few MPI processes / OpenMP threads as possible. Can you reproduce your problem with only one or two processes / threads?
- Get familiar with using TotalView by debugging a serial toy program first.

[70] http://www.totalviewtech.com/pdf/TV_DS_RE_x3.pdf
[71] http://www.totalviewtech.com/pdf/other/totalview_reference_guide.pdf

A.2.2 Debugging MPI Programs

A.2.2.1 Starting TotalView

- There are two ways to start the debugging of MPI programs: *New Launch* and *Classic Launch*.

  The *New Launch* is the easy and intuitive way to start a debugging session. Its disadvantage is the inability to detach from and reattach to running processes. Start TotalView as for serial debugging and use the *Parallel* pane in the *Startup Parameters* window to enable startup of parallel program run. The relevant items to adjust are the *Tasks* item (number of MPI processes to start) and the *Parallel System* item. The latter has to be set according to the MPI vendor used. Please use the *SUN MPI CT7* value for Oracle MPI (formerly Sun HPC Cluster Tools) 8.x - the *SUN MPI* value is for the old Cluster Tools version 6 which is not supported any more.

  The *Classic Launch*[72] helps to start a debug session from command line without any superfluous clicks in the GUI. It is possible to attach to a subset of processes and to detach/reattach again. The arguments that are to be added to the command line of **mpiexec** depend on the MPI vendor. For Intel MPI and Oracle MPI use the flag **-tv** to enable the Classic Launch:
  `$ $PSRC/pex/a20|| $MPIEXEC -tv -np 2 a.out < input`
  When the GUI appears, type *g* for go, or click *Go* in the TotalView window.

  TotalView may display a dialog box stating: *Process ... is a parallel job. Do you want to stop the job now?* Click **Yes** to open the TotalView debugger window with the source window and leave all processes in a traced state or **No** to run the parallel application directly.

You may switch to another MPI process by

- Clicking on another process in the root window
- Circulating through the attached processes with the **P-** or **P+** buttons in the process window

Open another process window by clicking on one of the attached processes in the root window with your right mouse button and selecting *Dive in New Window*.

A.2.2.2 Setting a Breakpoint By right-clicking on a breakpoint symbol you can specify its properties. A breakpoint will stop the whole process group (all MPI processes, default) or only one process. In case you want to synchronize all processes at this location, you have to change the breakpoint into a barrier by right clicking on a line number and selecting *Set Barrier* in the pull-down menu.

It is a good starting point to set and run into a barrier somewhere after the MPI initialization phase. After initially calling MPI_Comm_rank, the rank ID across the processes reveals whether the MPI startup went well. This can be done by right-clicking on the variable for the rank in the source pane, then selecting either Across Processes or Across Threads from the context menu.

A.2.2.3 Starting, Stopping and Restarting your Program You can perform stop, start, step, and examine single processes or groups of processes. Choose *Group* (default) or *Process* in the first pull-down menu of the toolbar.

[72] *Note*: The *Classic Launch* is known not to behave properly if using the Intel compiler with Oracle MPI. Until this bug is fixed, use the *New Launch*, or try to use another compiler-MPI combination.

A.2.2.4 Printing a Variable You can examine the values of variables of all MPI processes by selecting *View →Show Across →Processes* in a variable window, or alternatively by right-clicking on a variable and selecting *Across Processes*. The values of the variable will be shown in the array form and can be graphically visualized. One-dimensional arrays or array slices can also be shown across processes. The thread ID is interpreted as an additional dimension.

A.2.2.5 Message Queues You can look into outstanding message passing operations (unexpected messages, pending sends and receives) with the *Tools →Message Queue*. Use *Tools →Message Queue Graph* for visualization - you will see pending messages and communication patterns. Find deadlocks by selecting *Options →Cycle Detection* in an opened *Message Queue Graph* window.

A.2.3 Debugging OpenMP Programs

A.2.3.1 Some General Hints for Debugging OpenMP Programs Before debugging an OpenMP program, the corresponding serial program should run correctly. The typical OpenMP parallelization errors are data races, which are hard to detect in a debugging session because the timing behavior of the program is heavily influenced by debugging. You may want to use a thread-checking tool first (see chapter 7.4 on page 68).

Many compilers turn on optimization when using OpenMP by default. This default should be overwritten. Use e.g. the **-xopenmp=noopt** suboption for the Oracle compilers or **-openmp -O0** flags for the Intel compiler.

For the interpretation of the OpenMP directives, the original source program is transformed. The *parallel regions* are *outlined* into separate subroutines. *Shared* variables are passed as call parameters and *private* variables are defined locally. A parallel region cannot be entered stepwise, but only by running into a breakpoint. If you are using FORTRAN, check that the serial program does run correctly compiled with

- **-automatic** option (Intel **ifort** compiler) or
- **-stackvar** option (Oracle Studio **f95** compiler) or
- **-frecursive** option (GCC **gfortran** compiler) or
- **-Mrecursive** option (PGI **pgf90** compiler).

A.2.3.2 Compiling Some options, e.g. the ones for OpenMP support, cause certain compilers to turn on optimization. For example, the Oracle-specific compiler switches **-xopenmp** and **-xautopar** automatically invoke high optimization (**-xO3**).
Compile with **-g** to prepare the program for debugging and do not use optimization if possible:

- Intel compiler: use **-openmp -O0 -g** switches
- Oracle Studio compiler: use **-xopenmp=noopt -g** switches
- GCC compiler: use **-fopenmp -O0 -g** switches
- PGI compiler: use **-mp -Minfo=mp -O0 -g** switches

A.2.3.3 Starting TotalView Start debugging your OpenMP program after specifying the number of threads you want to use
`$ OMP_NUM_THREADS=`*`nthreads`*` totalview `*`a.out`*

The parallel regions of an OpenMP program are outlined into separate subroutines. Shared variables are passed as call parameters to the outlined routine and private variables are defined locally. A parallel region cannot be entered stepwise, but only by running into a breakpoint.

You may switch to another thread by

- clicking on another thread in the root window or
- circulating through the threads with the **T-** or **T+** buttons in the process window.

A.2.3.4 Setting a Breakpoint By right-clicking on a breakpoint symbol, you can specify its properties. A breakpoint will stop the whole process (group) by default or only the thread for which the breakpoint is defined. In case you want to synchronize all processes at this location, you have to change the breakpoint into a barrier by right-clicking on a line number and selecting *Set Barrier* in the pull-down menu.

A.2.3.5 Starting, Stopping and Restarting your Program You can perform stop, start, step, and examine single threads or the whole process (group). Choose *Group* (default) or *Process* or *Thread* in the first pull-down menu of the toolbar.

A.2.3.6 Printing a Variable You can examine the values of variables of all threads by selecting *View →Show Across →Threads* in a variable window, or alternatively by right-clicking on a variable and selecting *Across Threads*. The values of the variable will be shown in the array form and can be graphically visualized. One-dimensional arrays or array slices can be also shown across threads. The thread ID is interpreted as an additional dimension.

B Beginner's Introduction to the Linux HPC-Cluster

This chapter contains a short tutorial for new users about how to use the RWTH Aachen Linux HPC-Cluster. It will be explained how to set up the environment correctly in order to build a simple example program. Hopefully this can easily be adapted to your own code. In order to get more information on the steps performed you need to read the referenced chapters.

The first step you need to perform is logging in[73] to the HPC-Cluster.

B.1 Login

You must use the *secure shell protocol* (**ssh**) to log in. Therefore it might be necessary to install an ssh client on your local machine. If you are running Windows, please see chapter 4.1 on page 20 on where to get such an ssh client. Depending on the client you use, there are different ways to enter the necessary information. The name of the host you need to connect to is **cluster.rz.rwth-aachen.de** (other frontend nodes are named in table 1.1 on page 8) and your user name is usually your TIM ID.

On Unix or Linux systems, **ssh** is usually installed or at least included in the distribution. If this is the case you can open a console and enter the command

`$ ssh -Y <username>@cluster.rz.rwth-aachen.de`

You are then prompted for your password. After having entered it, you are logged in to the HPC-Cluster and see a shell prompt like this:

```
ab123456@cluster:~[1]$
```

The first word is your user name, in this case `ab123456`, separated by an "@" from the machine name `cluster`. After the colon the current directory is prompted, in this case ~ which is an alias for `/home/ab123456`. This is your home directory (for more information on available directories please see chapter 4.3 on page 22). Please note that your user name, contained in the path, is of course different from ab123456. The number in the brackets counts the entered commands. The prompt ends with the $ character. If you want to change your prompt, please take a look at chapter 4.4 on page 25.

You are now logged in to a Linux frontend machine.

The cluster consists of interactively accessible machines and machines that are only accessible by batch jobs. See chapter 4.5 on page 27. The interactive machines are not meant for time consuming jobs. Please keep in mind that there are other users on the system which are affected if the system gets overloaded.

B.2 The Example Collection

As a first step, we show you how to compile an example program from our Example Collection (chapter 1.3 on page 9). The Example Collection is located at `/rwthfs/rz/SW/HPC/examples`. This path is stored in the environment variable $PSRC.

To list the contents of the examples directory use the command *ls* with the content of that environment variable as the argument:

`$ ls $PSRC`

The examples differ in the parallelization paradigm used and the programming language which they are written in. Please refer to chapter 1.3 on page 9 or the README file for more information:

`$ less $PSRC/README.txt`

The examples need to be copied into your home directory (~) because the global directory is read-only. This is can be done using Makefiles contained in the example directories. Let's

[73] If you do not yet have an account for our cluster system you can create one in Tivoli Identity Manager (TIM): http://www.rz.rwth-aachen.de/tim

assume you want to run the example of a jacobi solver written in C++ and parallelized with OpenMP. Just do the following:

```
$ cd $PSRC/C++-omp-jacobi ; gmake cp
```

The example is copied into a subdirectory of your home directory and a new shell is started in that new subdirectory.

B.3 Compilation, Modules and Testing

Before you start compiling, you need to make sure that the environment is set up properly. Because of different and even contradicting needs regarding software, we offer the modules system to easily adapt the environment. All the installed software packages are available as modules that can be loaded and unloaded. The modules themselves are put into different categories to help you find the one you are looking for (see chapter 4.4.2 on page 25 for more detailed information).

Directly after login some modules are already loaded by default. You can list them with

```
$ module list
```

The output of this command looks like this:

```
Currently Loaded Modulefiles:
1) DEVELOP      2) intel/11.1   3) sunmpi/8.2
```

The default modules are in the category DEVELOP, which contains compilers, debuggers, MPI libraries etc., the Intel FORTRAN/C/C++ Compiler version 11.1 and Oracle (formerly Sun) MPI version 8.2.1. The list of available modules can be printed with

```
$ module avail
```

In this case, the command prints out the list of available modules in the DEVELOP category, because this category is loaded and the list of all other available categories. Let's assume that for some reason you'd like to use the GNU compiler instead of the Intel compiler for our C++/OpenMP example.[74]

All availble gcc versions can be listed by

```
$ module avail gcc
```

To use GCC version 4.4 do the following:

```
$ module switch intel gcc/4.4
```

Please observe how Oracle MPI is first unloaded, then loaded again. In fact, the loaded version of Oracle MPI is different from the unloaded version, because the loaded version is suitable for being used together with the GNU compiler whereas the unloaded is built to be used with the Intel compiler. The module system takes care of such dependencies.

Of course you can also load an additional module instead of replacing an already loaded one. For example, if you want to use a debugger, you can do a

```
$ module load totalview
```

In order to make the usage of different compilers easier and to be able to compile with the same command, several environment variables are set. You can look up the list of variables in chapter 5.2 on page 36.

Often, there is more than one step needed to build a program. The **make** tool offers a nice way do define these steps in a *Makefile*. We offer such Makefiles for the examples, which use the environment variables. Therefore when starting

```
$ gmake
```

the example will be built and executed according to the specified rules. Have a look at the Makefile if you are interested in more details.

[74] Usually, though, we'd recommend using the Intel or Oracle compilers for production because they offer better performance in most cases.

As the Makefile already does everything but explain the steps, the following paragraph will explain it step-by-step. You have to start with compiling the source files, in this case *main.cpp* and *jacobi.cpp*, with the C++ compiler:[75]

```
$ $CXX $FLAGS_DEBUG $FLAGS_FAST $FLAGS_OPENMP -DREAD_INPUT -c jacobi.cpp
main.cpp
```

This command invokes the C++ compiler stored in the environment variable $CXX, in this case *g++* as you are using the GNU compiler collection. The compiler reads both source files and puts out two object files, which contain machine code. The variables $FLAGS_DEBUG, $FLAGS_FAST, and $FLAGS_OPENMP contain compiler flags to, respectively, put debugging information into the object code, to optimize the code for high performance and to enable OpenMP parallelization. The *-D* option specifies C preprocessor directives to allow conditional compilation of parts of the source code. The command line above is equivalent to writing just the content of the variables:

```
$ g++ -g -O3 -ffast-math -mtune=opteron -fopenmp -DREAD_INPUT -c jacobi.cpp
main.cpp
```

You can print the values of variables with the *echo* command, which should print the line above

```
$ echo $CXX $FLAGS_DEBUG $FLAGS_FAST $FLAGS_OPENMP -DREAD_INPUT -c jacobi.cpp
main.cpp
```

After compiling the object files, you need to link them to an executable. You can use the linker *ld* directly, but it is recommended to let the compiler invoke the linker and add appropriate options e.g. to automatically link against the OpenMP library. You should therefore use the same compiler options for linking as you used for compiling. Otherwise the compiler may not generate all needed linker options. To link the objects to the program *jacobi.exe* you have to use

```
$ $CXX $FLAGS_DEBUG $FLAGS_FAST $FLAGS_OPENMP jacobi.o main.o -o jacobi.exe
```

Now, after having built the executable, you can run it. The example program is an iterative solver algorithm with built-in measurement of time and megaflops per second. Via the environment variable $OMP_NUM_THREADS you can specify the number of parallel threads with which the process is started. Because the *jacobi.exe* program needs input you have to supply an input file and start

```
$ export OMP_NUM_THREADS=1; ./jacobi.exe < input
```

After a few seconds you will get the output, including the runtime and megaflop rate, which depend on the load on the machine.

As you built a parallel OpenMP program it depends on the compiler with how many threads the program is executed if the environment variable $OMP_NUM_THREADS is not explicitly set. In the case of the GNU compiler the default is to use as many threads as processors are available.

As a next step, you can double the number of threads and run again:

```
$ export OMP_NUM_THREADS=2; ./jacobi.exe < input
```

Now the execution should have taken less time and the number of floating point operations per second should be about twice as high as before.

B.4 Computation in batch mode

After compiling the example and making sure it runs fine, you want to compute. However, the interactive nodes are not suited for larger computations. Therefore you can submit the example to the batch queue (for detailed information see chapter 4.5 on page 27). It will be

[75] If you are not using one of our cluster systems the values of the environment variables $CXX, $FLAGS_DEBUG et cetera are probably not set and you cannot use them. However, as every compiler has its own set of compiler flags, these variables make life a lot easier on our systems because you don't have to remember or look up all the flags for all the compilers and MPIs.

executed when a compute node is available.

You have to specify the command to run with *echo* and pipe the string to *qsub*, thus submitting the batch job. The **qsub** command needs several options in order to specify the required resources, e.g. the number of CPUs and amount of memory to reserve, the operating system and the runtime.

```
$ echo "module switch intel gcc/4.4; jacobi.exe < input" | qsub -o
`pwd`/output.txt -j y -l h_rt=00:01:00 -l h_vmem=700M -l threads=2 -l
ostype=linux -m beas -wd `pwd` -M <your_email_address>
```
[76]

You will get an email when the job is finished if you enter your email address instead of *<your_email_address>*. The output of the job will be written to ***output.txt*** in the current directory.

The same job can be scripted in a file, say simplejob.sh, in which the options of qsub are saved with "#$" magic cookie:

```
#!/usr/bin/sh
#$ -o output.txt
#$ -j y
#$ -cwd
#$ -l h_rt=00:01:00
#$ -l h_vmem=700M
#$ -l threads=2
#$ -l ostype=linux
#$ -m beas
#$ -M <your_email_address>

module switch intel gcc/4.4
jacobi.exe < input
```

To submit a job, use

```
$ qsub simplejob.sh
```

You can also mix both ways to define options; the options set over commandline are preferred.

[76] This is not the recommended way to submit jobs, however you do not need a job script here. You can find several example scripts in chapter 4.5 on page 27. The used options are explained there as well.

Index